

Computer Architecture and Organization

Design Principles and Applications

Second Edition

Author's Profile



Prof. B. Govindarajalu is presently Dean (Computer Studies) at Rajalakshmi Engineering College, Chennai. He is a B.E. in Electronics and Communication Engineering from Regional Engineering College, Trichirappali and M.Tech in Computer Science and Engineering from IIT Bombay. Before joining Rajalakshmi Engineering College he worked with Manipal Engineering College; IIT Bombay; ORG Systems (Baroda) Infotech Ltd. (Chennai); and Microcode (Chennai).

He has over 37 years of experience in computer hardware and IT industry covering design and development, manufacturing, maintenance, technical support, teaching and training. He has developed a series of microprocessor-based products such as the IBM System 360 Compatible Data Communications Multiplexer, VDU System Console and Terminal Processing Unit/Concentrator.

After 15 years of industrial experience, he founded Microcode, an IT firm specializing in PC hardware and networking training and development of diagnostic tools where he trained nearly 10,000 engineers in PC maintenance. He has developed a series of certification tests on computer architecture, PC hardware and networking. He has also taught personal computer hardware and architecture as visiting professor at several institutions such as Rajalakshmi Engineering College, Crescent Engineering College, Jaya Engineering College, Vellore Institute of Technology and Sastha Institute of Technology. He has conducted career courses on computer networking for Rajalakshmi Engineering College, Velammal Engineering College, Kongu Engineering College, JJ College of Engineering and SSN College of Engineering.

He has been a specialist for a UNESCO project on curriculum development. He has also served as a member of the Board of Studies at PSG College of Technology, Coimbatore.

He has authored two computer books published by Tata McGraw Hill:

1. IBM PC and Clones: Hardware, Troubleshooting and Maintenance
2. Computer Architecture and Organization: Design Principles and Applications

He is working on a book titled *Advanced Microprocessor Architectures*. His interests include Tamil literature, spirituality and meditation.

Govindarajalu's future plans include launching a technical magazine catering to the needs of engineering students. He may be contacted at bgrajulu@gmail.com.

Computer Architecture and Organization

Design Principles and Applications

Second Edition

B. Govindarajalu

(Founder-CEO, MICROCODE, Chennai),

Dean (Computer Studies)

Rajalakshmi Engineering College

Chennai



Tata McGraw-Hill Publishing Company Limited

NEW DELHI

McGraw-Hill Offices

New Delhi New York St Louis San Francisco Auckland Bogotá Caracas
Kuala Lumpur Lisbon London Madrid Mexico City Milan Montreal
San Juan Santiago Singapore Sydney Tokyo Toronto



Tata McGraw-Hill

Published by the Tata McGraw Hill Education Private Limited,
7 West Patel Nagar, New Delhi 110 008.

Copyright © 2010, 2009, 2007, 2006, 2005 by Tata McGraw Hill Education Private Limited.

No part of this publication may be reproduced or distributed in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise or stored in a database or retrieval system without the prior written permission of the publishers. The program listings (if any) may be entered, stored and executed in a computer system, but they may not be reproduced for publication.

This edition can be exported from India only by the publishers,
Tata McGraw Hill Education Private Limited.

ISBN (13): 978-0-07-015277-9

ISBN (10): 0-07-015277-2

Vice President and Managing Director—Asia-Pacific Region: *Ajay Shukla*

Executive Publisher—Professional: *R Chandra Sekhar*

Production Executive: *Rita Sarkar*

Manager—Sales and Marketing: *S Girish*

Deputy Marketing Manager—Science, Technology & Computing: *Rekha Dhyani*

General Manager—Production: *Rajender P Ghansela*

Assistant General Manager—Production: *B L Dogra*

Information contained in this work has been obtained by Tata McGraw-Hill, from sources believed to be reliable. However, neither Tata McGraw-Hill nor its authors guarantee the accuracy or completeness of any information published herein, and neither Tata McGraw-Hill nor its authors shall be responsible for any errors, omissions, or damages arising out of use of this information. This work is published with the understanding that Tata McGraw-Hill and its authors are supplying information but are not attempting to render engineering or other professional services. If such services are required, the assistance of an appropriate professional should be sought.

Typeset at Script Makers, 19, A1-B, DDA Market, Paschim Vihar, New Delhi 110 063, and printed at Sai Printo Pack, A-102/4, Okhla Industrial Area, Phase II, New Delhi 110020.

Cover Design: Kapil Gupta

Cover Printer: Sai Printo Pack

RAZYCRBZDXQYY

To
my in-laws
Mr. N. Gopalakrishnan, my father-in-law and Mrs. Leelavathy Gopalakrishnan
who have always encouraged and supported me
in all my new ventures
and
my recently discovered friend
Mr. G.R. Krishnamoorthy (EX OM, CCS Ltd. Chittoor)
whose simplicity, honesty and hardworking nature
have deeply impressed me

Foreword I

I am extremely happy to write the Foreword to the second edition of B. Govindarajalu's book on *Computer Architecture Organization: Design Principles and Applications*. His earlier book, *IBM PC and Clones: Hardware, Troubleshooting and Maintenance*, was well received—not only in academic circles but also in the computer maintenance industry.

The first edition of this book was highly appreciated and lays emphasis upon learning of fundamental concepts through easily understandable text. Good selection of problems with their solutions helps students to grasp the intricacies of the subject and instill a sense of self study, creative thinking and review.

The second edition of this book covers important units of a computer in a very comprehensible manner and therefore, will be useful as a Level I book for undergraduate students pursuing courses in electrical engineering, electronics, computers and IT. All the chapters are concise and are supported by a detailed background. The author's vast experience and sound theoretical and practical knowledge is reflected in the material covered. This book will serve as a valuable guide for students keen on mastering the organization of computer architecture.

Prof. S.S.S.P. Rao

*Chief Mentor and Advisor, CMC Limited,
Hyderabad*

*Former Professor of Department of Computer Science and Engineering,
IIT Bombay*

*Visiting Professor of Department of Computer Science and Engineering,
IIT Hyderabad*

Foreword II

There are many good books which cover the vast and ever-growing area of Computer Architecture but not all cover the entire gamut of emerging technologies in this domain. Also, at times the coverage as well as language do not suit Indian academic community.

Computer Architecture and Organization: Design Principles and Applications written by one of my colleagues, during my earlier stint, has addressed the issues lucidly and presented the technical content relevant to most of the Indian universities.

In the second edition, few additional chapters covering advanced topics such as Parallelism and Super Scalar Architecture have been included.

The author who has rich experience both in academia and industry has ensured that the overall pedagogic content is easy to follow and comprehend. I recommend this book for teachers, university students and professionals.

Dr. K. Sarukesi

Vice-Chancellor

*Hindustan Institute of Technology and Science,
Chennai*

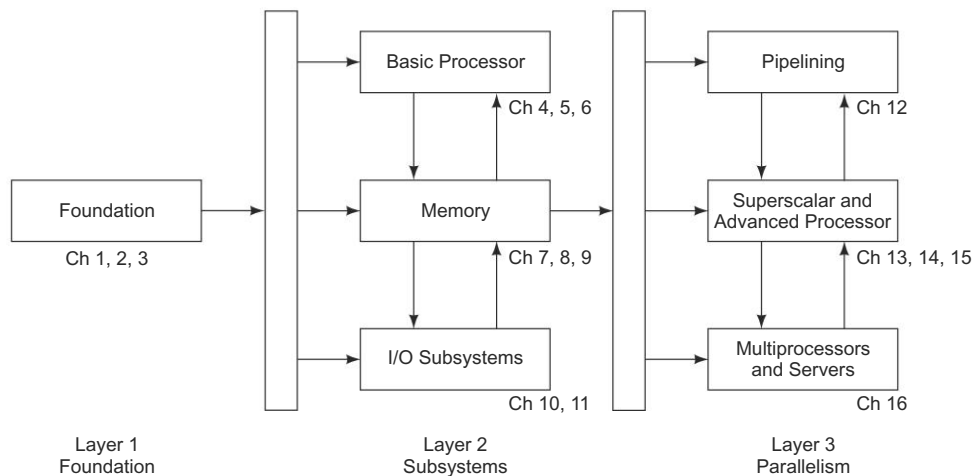
Preface to the Second Edition

Computer design or computer architecture is an interesting subject. With the rapid evolution of computer science, an increasingly diverse group of students is studying the subject of computer architecture. Though interesting and simple a majority of students consider it to be highly complex. Having worked both in industry and academic institutions with a range of computers starting from second generation systems, I decided to help in such a way that even a student of average IQ can understand it. The target audience of this book includes readers who want to learn basic computer organization as well as those who want to design a computer.

It can also be used by professionals as a reference. The book is intended as a text book for undergraduate students of computer science, information technology and electrical and electronics engineering. This book serves as a first level course on computer architecture/organization. As a prerequisite, the reader is expected to be familiar with computer programming. It is assumed that the reader does not know any specific programming language. In addition, exposure to digital electronics is helpful. For the benefit of those who have no knowledge of this, an annexure presents an overview of essential topics of digital electronics.

ORGANIZATION OF THE BOOK

This book is organized with a three layer structure shown here.



Though the chapters are organized in a linear sequence for reading, alternate layouts are possible. The first three chapters form the top layer that provides a foundation to computer architecture and organization. After studying these, the remaining chapters can be covered in any sequence shown in the figure. The middle layer consisting of chapters 4–11 describes the design of different subsystems of the computer. The last layer covers special topics that deal with parallelism and advanced computer architectures. This layer comprises chapters 12–16. This modular arrangement facilitates quick and standalone reference by professionals for select topics.

CHAPTER DESCRIPTION

Layer 1: Foundation

Chapter 1 introduces the basics of a present-day computer without describing the design aspects. It defines the computer's hardware and software layers and provides an overview of Interrupt Concept and I/O techniques. Chapter 1 also introduces techniques used in high performance computers in addition to system performance measurement.

Chapter 2 gives an overview of the history of computer and its evolution. Generations of computers are defined along with the new features of each generation.

Chapter 3 describes the basic attributes of a computer such as instruction set, addressing modes and data types. It also distinguishes between RISC and CISC architectures.

Layer 2: Subsystems

Chapter 4 focuses on data path organization and ALU design. Both fixed-point data path and floating-point data path are covered.

Chapter 5 describes the algorithms for fixed-point and floating-point arithmetic. It also defines serial and parallel adders.

Chapter 6 presents techniques of designing a control unit. Both hardwired and micro-programmed control units are covered. It also describes micro operations and register transfer language. Single bus processor as well as multiple bus processor are discussed.

Chapter 7 focuses on memory technologies and main memory design.

Chapter 8 describes various memory enhancement techniques for tackling performance and capacity issues. Cache memory, virtual memory and memory interleaving are the major topics covered.

Chapter 9 deals with secondary storage devices such as magnetic disks and tapes apart from optical media. In addition, RAID levels and RAID controllers are discussed.

Chapter 10 presents the sequence of communication between the internal units. Different I/O controllers, bus standards, Interrupt concept and I/O techniques are described.

Chapter 11 describes the commonly used peripheral devices.

Layer 3: Parallelism and Advanced Architectures

Chapter 12 gives an elaborate study of parallelism and concurrency in computers with special emphasis on pipelining. The concept of RISC processors is also covered here.

Chapters 13–16 describe advanced architectures found in uniprocessing and multiprocessing. Chapter 13 deals with the superscalar architecture of uniprocessors. Dynamic scheduling techniques are discussed.

Chapter 14 describes VLIW and EPIC architectures.

Chapter 15 deals with vector computing and array processing.

Chapter 16 describes multiprocessors and cache coherency in addition to reliability and fault tolerance concepts and techniques.

Additions in Second Edition

The Second Edition aims to:

- Enhance the suitability of the book according to the syllabi of the UG course on Computer Organization and Architecture offered by most universities, especially Anna University, Chennai.
- Update the technical content according to the changes in the industry since the publication of the first edition in 2003.
- Improve quality.

Two major changes have been carried out in the Second Edition:

- Chapters 11 and 12 of the first edition have been completely revised and reorganized in the second edition as five different chapters (12 to 16).
- Part of the contents of Chapter 10 in the first edition, dealing with magnetic disk, tape and optical disk, have been shifted to Chapter 11 in the second edition.

In addition, several minor additions have been made to the contents in almost all the chapters. Some of these are identified in the following table:

Chapter no.	Topics added/modified	Relevant section
1	System Performance Measurement and Response Time	1.13
3	Hardware-Software Interface	3.8
4	Floating-point Numbers	4.4
5	Building Long Adder	5.3
6	Synchronous and Asynchronous Control Unit	6.7
6	Clocking and Synchronization	6.3
6	Single Cycle and Multicycle Design	6.3
6	Single Bus and Multiple Bus Processor	6.4, 6.5
9	End-to-End Data Protection	9.6.8
9	RAID	9.7
9	Magnetic Tape	9.8
10	PCI Bus	10.8
10	SCSI	10.8

(Contd.)

(Continued)

Chapter no.	Topics added/modified	Relevant section
10	USB	10.8
11	Laser Printer	11.4
11	Inkjet Printer	11.4
12	RISC Pipeline	12.8
12	Operand Forwarding	12.9
12	Exception Handling	12.12
13	Loop Unrolling	13.7
13	Scoreboard	13.9
13	Tomasulo Algorithm	13.10

I welcome comments and suggestions from the readers for improvement of the book.

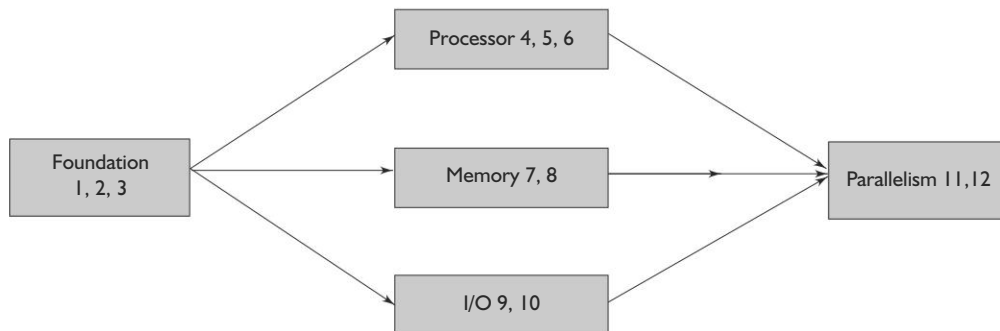
B. GOVINDARAJALU
bgrajulu@gmail.com

Preface to the First Edition

Today computers find application in all spheres of life. And an increasingly diverse group of students are studying the subject. Though an interesting and simple subject, a majority of students known to me consider it highly complex, consisting of complicated and advanced topics. As I have worked with a range of computers, from the second generation to the modern systems both in the industry and the academia, I felt a need to provide a helping hand to such students. My aim is to simplify the subject and make it easy even to an average student.

This book serves as a first level course on computer architecture and organization. As a pre-requisite, the reader is expected to be familiar with computer programming. In addition to this, exposure to digital electronics will be helpful. For the benefit of those who have no prior knowledge of this, the book includes an overview of the essential topics of digital electronics in Annexure 3.

This book is organized in five modules with a three-layer structure shown in the following figure.



The first three chapters form the top layer that provides a base to computer architecture and organization. After studying this, the remaining chapters can be covered in any of the three different paths shown in the figure. The middle layer, consisting of Chapters 4–10, describes the design of the different sub-systems of the computer. The last layer, consisting of Chapters 11 and 12, covers special topics that deal with parallelism and high performance of advanced computer architectures. This modular arrangement facilitates quick and stand-alone reference of selected topics by the professionals.

Chapter 1 introduces the basics of the present day computer without dwelling on the design aspects. It defines the hardware and software layers in a computer and provides an overview of interrupt concept and I/O techniques.

Chapter 2 gives an overview of the history of computers and their evolution and defines the different generations of computers.

Chapter 3 describes the basic attributes of a computer such as instruction set, addressing modes and data types. It also distinguishes between RISC and CISC architectures.

Chapter 4 describes the algorithms for fixed-point and floating-point arithmetic. It also defines serial and parallel adders.

Chapter 5 focuses on the datapath organization and ALU design. It also describes micro-operations and register transfer language.

Chapter 6 presents the techniques of designing a control unit. Both hardwired and microprogrammed control unit are covered.

Chapter 7 focuses on the memory technologies and main memory design.

Chapter 8 describes the various memory enhancement techniques for tackling performance and capacity issues. Cache memory, virtual memory and memory interleaving are the major topics covered.

Chapter 9 presents the sequence of communication between the internal units. Different I/O controllers, bus standards, interrupt mechanism and I/O techniques are described.

Chapter 10 describes the commonly used peripheral devices.

Chapter 11 gives an elaborate study of parallelism and concurrency in computers, with a special emphasis on pipelining and vector computing.

Chapter 12 describes advanced architectures found in uni-processing and multi-processing. RISC, Superscalar and VLIW architectures of uni-processor are covered. Multi-processors and cache coherency are also discussed, in addition to reliability and fault tolerance concepts and techniques.

I look forward to the comments and suggestions from the readers for further improving the book.

B. GOVINDARAJALU

Acknowledgements

When my first book *IBM PC and Clones: Hardware, Troubleshooting and Maintenance* was completed in 1991, I had no idea that I would be writing a book on Computer Architecture. The need for this book was felt during the special coaching I conducted in 2002 for a small group of students of Anna University, Chennai. I thank all my students of the past 30 years whose interactions have motivated me constantly to keep abreast of computer architecture and hardware concepts. I would like to extend special thanks to Ms Jamuna Maruthachalam who along with her classmates formed the pilot batch for this book.

I believe that my ability to author a book of this genre came from the opportunities and experiences I had in my career. To a large extent, my stay at IIT Bombay, both as a student and as an employee, had a strong impact on my technical and academic skills. I also thank Prof. James R Isaac and Dr. SSSP Rao, whose guidance from 1973 to 1979, at IIT Bombay, helped lay down the basic foundation for this book.

I sincerely thank our college management, especially Shri S. Meganathan, correspondent and Dr.(Mrs.) Thangam Meganathan, chairperson, for providing excellent infrastructure and environment in Rajalakshmi Engineering College that was instrumental in successfully bringing out the second edition.

I thank my ex-colleagues at Microcode (while work was in progress on the first edition) and present colleagues at Rajalakshmi Engineering College who supported me during the course of authoring this book.

I am also thankful to the Professional team of Tata McGraw Hill, for their encouragement and support, right from conception to publishing of this book in a short span of time.

Writing a book requires extensive support from family. I thank my son Krishnakumar, and my daughter Padma Ananth, they reviewed certain portions and offered their comments. I also thank my wife Bhuvaneswari, and my daughter-in-law Manjula Krishnakumar, who assisted me in several ways, especially while I was preparing the figures. Special thanks to my grand son Vihaan, who frequently hindered progress of the book by his mischievous antics. It was he who provided unlimited entertainment and proved to be a welcome distraction whenever I needed it while authoring the second edition.

B. GOVINDARAJALU

Contents

<i>Foreward I</i>	<i>vii</i>
<i>Foreward II</i>	<i>ix</i>
<i>Preface to the Second Edition</i>	<i>xi</i>
<i>Preface to the First Edition</i>	<i>xv</i>
<i>Acknowledgements</i>	<i>xvii</i>
<i>Abbreviations</i>	<i>xxvii</i>

1. Basic Computer Organization **1**

1.1	Introduction	3
1.2	Man and Computing	3
1.3	Digital Computer	5
1.4	Computer Organization and Functional Units	9
1.5	Main Memory	16
1.6	CPU Operational Concepts	22
1.7	Interrupt Concept	28
1.8	I/O Techniques	29
1.9	Bus Concept	31
1.10	System Software	35
1.11	Computer Types	36
1.12	Computer Performance Factors	39
1.13	System Performance Measurement	40
1.14	High Performance Techniques	43
1.15	Computer Design Process	49
1.16	Computer Structure	50
1.17	Computer Function	53
1.18	Architecture and Organization	55
	SUMMARY	56
	REVIEW QUESTIONS	58
	EXERCISES	59

2. Evolution of Computers **61**

2.1	Introduction	63
2.2	Dimensions of Computer Evolution	63

- 2.3 History of Computers 71
- SUMMARY 85
- REVIEW QUESTIONS 85

3. CPU Architecture and Instruction Set Principles 87

- 3.1 Introduction 89
- 3.2 Processor Role 89
- 3.3 Processor Design Goals 97
- 3.4 Processor Design Process 93
- 3.5 Datapath Organization 98
- 3.6 Control Unit Organization 98
- 3.7 Instruction Sequencing 99
- 3.8 Hardware–Software Interface 103
- 3.9 CISC vs RISC 105
- 3.10 Instruction Set Design 108
- 3.11 Instruction Types and Operations 118
- 3.12 Addressing Modes 123
- 3.13 Data Representation 130
- 3.14 Binary Data 131
- SUMMARY 132
- REVIEW QUESTIONS 134
- EXERCISES 134

4. ALU Design and Datapath 137

- 4.1 Introduction 139
- 4.2 Arithmetic Types 139
- 4.3 Fixed-Point Numbers 140
- 4.4 Floating-Point Numbers 143
- 4.5 Fixed-Point Datapath 147
- 4.6 Design of Arithmetic Unit 153
- 4.7 Design of Logic Unit 154
- 4.8 Design of Shifter 155
- 4.9 ALU Design 157
- 4.10 Typical Minicomputer Datapath 159
- 4.11 Typical Mainframe Datapath 161
- 4.12 Main Memory Interface 162
- 4.13 Local Storage/Register File 163
- 4.14 Datapath for Simple Instructions 163

- 4.15 Floating-Point Unit Datapath 169
- 4.16 Advanced Processors and Datapaths 173
 - SUMMARY 173
 - REVIEW QUESTIONS 174
 - EXERCISES 175

5. Binary Arithmetic Operations 177

- 5.1 Introduction 179
- 5.2 Hardware Components 179
- 5.3 Fixed-Point Arithmetic 188
- 5.4 Floating-Point Arithmetic 225
 - SUMMARY 235
 - REVIEW QUESTIONS 235
 - EXERCISES 236

6. Instruction Processing and Control Unit 237

- 6.1 Introduction 239
- 6.2 Basic Processing Concepts 239
- 6.3 Instruction Cycle and Processor Actions 244
- 6.4 Single-bus Processor 252
- 6.5 Multiple-bus Processor 264
- 6.6 Control Unit Design Options 267
- 6.7 Hardwired Control Unit 267
- 6.8 Microprogrammed Control Unit 276
 - SUMMARY 293
 - REVIEW QUESTIONS 294
 - EXERCISES 295

7. Main Memory Design 297

- 7.1 Introduction 299
- 7.2 Memory Parameters 299
- 7.3 Classification of Memory 301
- 7.4 ROM Subsystem Design 307
- 7.5 Static RAM IC 307
- 7.6 Dynamic RAM 313
- 7.7 Main Memory Allocation 327
- 7.8 Memory Hierarchy and Performance 329

SUMMARY	331
REVIEW QUESTIONS	332
EXERCISES	332

8. Memory Management Techniques **333**

8.1	Introduction	335
8.2	Main Memory Drawbacks	335
8.3	Instruction Prefetch	336
8.4	Memory Interleaving	338
8.5	Write Buffer	341
8.6	Cache Memory	342
8.7	Virtual Memory	368
8.8	Associative Memory	378
	SUMMARY	380
	REVIEW QUESTIONS	382
	EXERCISES	383

9. Secondary Storage Devices **385**

9.1	Introduction	387
9.2	Magnetic Storage Devices	387
9.3	Basic Principle of Magnetic Disk	388
9.4	Floppy Diskette	393
9.5	Overall Operation of Floppy Disk Subsystem	400
9.6	Hard Disk Drive (HDD)	404
9.7	RAID / DISK ARRAYS	431
9.8	Magnetic Tape Data Storage	437
9.9	Optical Disk Storage	446
	SUMMARY	451
	EXERCISES	451

10. I/O Concepts and Techniques **453**

10.1	Introduction	455
10.2	Accessing I/O Devices	455
10.3	Interrupt Handling	466
10.4	I/O Techniques in PC	480
10.5	Bus Arbitration Techniques	496
10.6	Bus Concepts	500
10.7	Interface Circuits and I/O Controllers	528
10.8	Modern Standard I/O Interfaces	556

SUMMARY	596
REVIEW QUESTIONS	598
EXERCISES	599

11. I/O Devices **601**

11.1	Introduction	603
11.2	Keyboard	603
11.3	CRT Display Monitor	608
11.4	Printer	613
11.5	Special Types of Disk Drives	625
11.6	Mouse and Trackball	627
11.7	Modem	628
11.8	Scanner	629
11.9	Digital Camera	630
11.10	Special Peripherals	630
	SUMMARY	637
	REVIEW QUESTIONS	637
	EXERCISES	637

12. RISC, Parallelism and Pipelining **639**

12.1	Introduction	641
12.2	RISC Systems and CISC Drawbacks	641
12.3	Techniques of RISC Systems	643
12.4	RISC Architecture in Modern Processors	646
12.5	Performance Enhancement Strategies	648
12.6	Classification of Parallelism	651
12.7	Multiple Functional Units	655
12.8	Pipelining	657
12.9	Pipeline Hazards	668
12.10	Influence on Instruction Sets	683
12.11	Data Path and Control Considerations	684
12.12	Exception Handling	684
	SUMMARY	685
	REVIEW QUESTIONS	686
	EXERCISES	687

13. Superscalar Architecture **689**

13.1	Introduction	691
13.2	Concept of Superscalar Architecture	691

13.3	Case Study 1—Intel Pentium Processor	696
13.4	Out-of-Order Execution	700
13.5	Register Renaming	701
13.6	Speculative Execution and Branch Prediction	704
13.7	Loop Unrolling	705
13.8	Dynamic Scheduling and Static Scheduling	707
13.9	Thornton Technique and Scoreboard	710
13.10	Tomasulo Algorithm and Reservation Stations	717
13.11	Case Study 2—Intel Pentium-pro Processor	730
13.12	NetBurst Architecture and Pentium 4	736
13.13	The Alpha Family	738
13.14	PowerPC	739
13.15	SPARC	740
	SUMMARY	741

14. VLIW and EPIC Architecture **743**

14.1	Introduction	745
14.2	VLIW Architecture	745
14.3	VLIW Compiler Techniques	746
14.4	VLIW vs Superscalar Processor	749
14.5	Tree Instruction	751
14.6	EPIC Architecture	752
	SUMMARY	755

15. Vector Computing and Array Processing **757**

15.1	Introduction	759
15.2	Vector Computing	759
15.3	Array Processor	769
	SUMMARY	770

16. Multiprocessor Systems and Servers **773**

16.1	Introduction	775
16.2	Classification of Multi-processors	775
16.3	Symmetric Multi-Processor (SMP)	777
16.4	Interconnection Structures	778
16.5	Clusters	782
16.6	Cache Coherence Problem	783
16.7	Fault Tolerance	786

16.8	Server Systems	792
	SUMMARY	797

Annexure 1 Number Systems and Conversions 799

A1.1	Introduction	799
A1.2	Common Number Systems	799
A1.3	Decimal Number System	801
A1.4	Conversion of Decimal Numbers into Other Number Systems	801
A1.5	Conversion of Other Systems into Decimal Number System	803

Annexure 2 Types of Computer Software 804

A2.1	Introduction	804
A2.2	System Software Types	804
A2.3	Programming Software	805
A2.4	Application Software	805

Annexure 3 ASCII Code 809

Annexure 4 Hardware Components and ICs 818

A4.1	Introduction	818
A4.2	Hardware Components: Discrete and Integrated	818
A4.3	Pulse Circuits and Waveforms	819
A4.4	Positive and Negative Logic	821
A4.5	High Active and Low Active Signals	821
A4.6	Combinational and Sequential Circuits	822
A4.7	Simple Gates	822
A4.8	Special Gates	823
A4.9	IC Families	824
A4.10	TTL Characteristics	827
A4.11	Open Collector TTL	828
A4.12	Tri-State Gates	829
A4.13	MOS Families	830
A4.14	Interfacing Logic Families	831
A4.15	Latch	832
A4.16	Flip-Flop	834
A4.17	Register	838

A4.18 Shift Register	838
A4.19 Counters	839
A4.20 Multiplexer	842
A4.21 Demultiplexer	844
A4.22 Decoder	845
A4.23 Encoder	846
A4.24 74S280 Parity Generator/Checker	847

Index

849

ABBREVIATIONS

μR	Micro Routine
μs	Micro second
ns	Nano second
AC	Accumulator
ACC	Accumulator
ACK	Acknowledgment
ADDR SEL	Address Select
AGU	Address Generation Unit
AHOLD	Address Hold
ALU	Arithmetic and Logic Unit
AOI	And-Or-Inverter
AR	Address Register
ASCII	American Standard Code for Information Interchange
ASIC	Application Specific Integrated Circuit
AT	Advanced Technology
ATC	Advanced Transfer Cache
b	Bit
B	Byte
BA	Branch Address
BAL	Bus Arbitration Logic
BCD	Binary Coded Decimal
BCZ	Byte Count Zero
BG	Bus Grant
BIOS	Basic Input-Output Control System
BIST	Built-in Self Test
BOT	Beginning Of Tape
BPI	Bits Per Inch
BPL	Branch Prediction Logic
BR	Bus Request
BTB	Branch Target Buffer

xxviii *Abbreviation*

BUN	Branch Unconditionally
BZ	Branch If Zero
C	Carry
CAD	Computer Aided Design
CAM	Content Address Memory
CAS	Column Address Strobe
CAW	Channel Address Word
CCD	Charge Coupled Device
CCW	Channel Command Word
CD	Compact Disc
CDB	Common Data Bus
CD-R	Compact Disc Recordable
CDRAM	Cache DRAM
CD-ROM	Compact Disc Read Only Memory
CD-RW	Compact Disc Rewritable
CISC	Complex Instruction Set Computing
CLA	Carry Look-Ahead Adder
CLK	Clock
CLV	Constant Linear Velocity
CM	Control Memory
CMAR	Control Memory Address Register
CMDR	Control Memory Data Register
CPI	Cycles Per Instruction
CPS	Characters Per Second
CPU	Central Processing Unit
CR	Carriage Return/Control Register
CRC	Cyclic Redundancy Check
CRCC	Cyclic Redundancy Check Character
CRT	Cathode Ray Tube
CRTC	CRT Controller
CU	Control Unit
CWP	Common Window Printer
DAT	Digital Audio Tape
DCB	Device Control Block/Device Command Block
DCE	Data Communication Equipment
DDR SDRAM	Double Data Rate Synchronous DRAM
DEC	Digital Equipment Corporation

DI	Disable Interrupt
DIB	Dual Independent Bus
DIMM	Dual In-line Memory Module
DMA	Direct Memory Access
DMA ACK	DMA Acknowledgement
DMA REQ	DMA Request
DMP	Dot Matrix Printer
DPL	Deep Pipeline Latency
DPU	Data Processing Unit
DR	Dynamic Reconfiguration/Data Register
DRAM	Dynamic Random Access Memory
DRQ	Data Request
DSDD	Double-sided Double Density
DSHD	Double-sided High Density
DSM	Distributed Shared Memory
DSQD	Double-sided Quad Density
DTE	Data Terminal Equipment
DTP	Desk Top Publishing
DVD	Digital Versatile Disk
EADS	Valid External Address
EAROM	Electrically Alterable ROM
EBCDIC	Extended Binary Coded Decimal Interchange Code
ECC	Error Checking and Correcting Code
ECP	Extended Capability Port
EDC	Electronic Digital Computer
EDP	End-to-end Data Protection
EDVAC	Electronic Discrete Variable Computer
EEPROM	Electrically Erase Programmable ROM
EI	Enable Interrupt
EIDE	Enhanced IDE
EISA	Extended Industry Standard Architecture
ENIAC	Electronic Numeric Indicator and Computer
EO	Execute Operation
EPIC	Explicitly Parallel Instruction Computing
EPP	Enhanced Parallel Port
EPROM	Erase Programmable Read Only Memory
ESC	Escape Sequence

XXX *Abbreviation*

EU	Execution unit
f	Frequency
FA	Full Adder
FDC	Floppy Disk Controller
FDD	Floppy Disk Drive
FF	Form Feed
FIFO	First-In First-Out
FM	Frequency Modulation
FPM	Fast Page Mode
FRC	Functional Redundancy Check
FSK	Frequency Shift Key
FU	Fetch unit
GMR	Giant Magneto Resistance
GPG	Graphics Performance Characterization Group
GPR	General Purpose Register
GPU	Graphics Processing Unit
GT	Grant
GUI	Graphical User interface
HA	Half Adder
HDA	Head Disc Assembly
HDC	Hard Disk Controller
HDD	Hard Disk Drive
Hexa	Hexa Decimal
HLL	High Level Language
HLT	Halt
HMT	Hardware Multi-Threading
HPG	High Performance Group
HSYNC	Horizontal Synchronization
Hz	Hertz
I/O	Input/Output
IAS	Institute for Advanced Studies
IB	Instruction Buffer
IBM	International Business Machines
IBR	Instruction Buffer Register
IC	Integrated Circuit
ID	Identifier/Instruction Decode
IDE	Integrated Drive Electronics

IE	Interrupt Enable
IEEE	Institute of Electrical and Electronics Engineers
IF	Instruction Fetch
ILP	Instruction Level Parallelism
IN	Input
INC	Increment
INTA	Interrupt Acknowledgement
INTR	Interrupt Request
IOAR	I/O Address Register
IODR	I/O Data Register
IOR	I/O Read
IOW	I/O Write
IR	Instruction Register
IRET	Return from Interrupt
IRG	Inter-record Gap
IRQ	Interrupt Request
ISA	Industry Standard Architecture
ISR	Interrupt Service Routine
IU	Instruction Unit
KB	Kilo Byte
KEN	Cache Enable
KHz	Kilo Hertz
KISS	Keep It Short and Simple
LAN	Local Area Network
LCD	Liquid Crystal Display
LDA	Load Accumulator
LED	Light Emitting Diode
LF	Line Feed
LFU	Least Frequently Used
LM	Local Memory
LMSW	Load Machine Status Word
LPM	Lines Per Minute
LQP	Letter Quality Printer
LRU	Least Recently Used
LSAR	Local Storage Address Register
LSB	Least Significant Bit
LSD	Least Significant Digit

LSDR	Local Storage Data Register
LSH	Least Significant Half
LSI	Large Scale Integration
LSR/W	Local Storage Read / Write Flag
MA	Memory Address
MAR	Memory Address Register
MB	Mega Byte
MBR	Memory Buffer Register
MCMAR	Microprogram Control Memory Address Register
MD	Multiplicand
MDR	Memory Data Register
MEMR	Memory Read
MEMW	Memory Write
MFC	Memory Function Complete
MFM	Modified Frequency Modulation
MHz	Mega Hertz
MIG	Metal In Gap
MIMD	Multiple Instruction Stream, Multiple Data Stream
MIPS	Millions of Instructions Per Second
MISD	Multiple Instruction Stream, Single Data Stream
ML	Machine Language
MM	Main Memory
MMU	Memory Management Unit
MMX	Multi-media Extension
MOB	Memory Order Buffer
MODEM	Modulator cum Demodulator
MQ	Multiplier-Quotient
MR	Memory Read/Magneto Resistance
ms	Milli Second
MSB	Most Significant Bit
MSH	Most Significant Half
MSI	Medium Scale Integration
MTBF	Mean Time Between Failures
MTTR	Mean Time To Repair
MUX	Multiplexor
MW	Memory Write
NA	Next Address

NAK	Negative Acknowledgement
NAS	Network Attached Storage
NCMAR	Nanoprogram Control Memory Address Register
NDP	Numeric Data Processor
NLQ	Near Letter Quality
NMA	Next Microinstruction Address
NMI	Non Maskable Interrupt
NNA	Next Nanoinstruction Address
NOOP	No Operation
NUMA	Non-uniform Memory Access
OAC	Operand Address Calculation
OCR	Operation Code Register
OF	Operand Fetch
OOO	Out-Of-Order Execution
OS	Operating System
OSG	Open System Group
OUT	Output
PC	Personal Computer (or) Program Counter
PCB	Printed Circuit Board
PCI	Peripheral Component Interconnect
PCU	Program Control Unit
PD	Pre Decode
PF	Pre Fetch
PIC	Programmable Interrupt Controller
PLL	Phase Lock Loop
PMR	Perpendicular Magnetic Recording
PnP	Plug and Play
POE	Plan Of Execution
POR	Power-On Reset
POST	Power-On Self-Test
PPD	Parallel Presence Detect
PPM	Pages Per Minute
PR	Processor
PROM	Programmable Read Only Memory
PS	Program Status
PSK	Phase Shift Key
PSW	Program Status Word

Q	Quotient
QIC	Quarter Inch
QS	Queue Status
RA	Register
RAID	Redundant Array of Independent Disks
RAM	Random Access Memory (or) Read/Write memory
RAMDAC	Random Access Memory Video Digital-to-Analog Converter
RAS	Reliability, Availability and Serviceability / Row Address Strobe
RAT	Register Alias Table
RAW	Read After Write
Rclk	Receive Clock
RDAT	Rotary Head Digital Audio Tape
RDRAM	Rambus DRAM
RISC	Reduced Instruction Set Computing
RLL	Run Length Limited
ROM	Read Only Memory
RQ	Request
RRF	Retirement Register File
RTL	Register Transfer Language (or) Register Transfer Level
RV	Reset Vector
RWM	Read Write Memory
S	Sign/Sum
SAN	Storage Area Network
SATA	Serial ATA
SC	Sequence Counter
SCSI	Small Computer System Interface
SDRAM	Synchronous Dynamic RAM
SDT	Segment Descriptor Table
SEL	Select
SIMD	Single Instruction stream, Multiple Data stream
SIMM	Single Inline Memory Module
SIN	Serial In
SIPO	Serial In Parallel Out
SIS	Start Interrupt Service
SISD	Single Instruction Stream, Single Data Stream
SLSA	Start Local Storage Access
SMI	System Management Interrupt

SMMA	Start Main Memory Access
SMP	Symmetric Multiprocessor
SOHO	Small Office Home Office
SOUT	Serial Out
SP	Stack Pointer
SPARC	Scalable Processor Architecture
SPD	Serial Presence Detect
SPEC	System Performance Evaluation Corporation
SPM	Scratch-Pad Memory
SPP	Standard Parallel Port
SR	Store Result
SRAM	Static Random Access Memory
SSE2	Streaming SIMD Extensions 2
SSI	Small Scale Integration
STA	Store Accumulator
SUB	Subtract
T	Time Period / Temporary Register
TB	Tera Byte
TC	Terminal Count
Tclk	Transmit Clock
TLB	Translation Look Aside Buffer
TPI	Tracks Per Inch
Tpw	Pulse width
UART	Universal Asynchronous Receiver Transmitter
UMA	Uniform Memory Access
UNIVAC	Universal Automatic Computer
USB	Universal Serial Bus
VCD	Video CD
VESA	Video Electronics Standard Association
VGA	Video Graphics Array
VLIW	Very Long Instruction Word
VLSI	Very Large Scale Integration
VRAM	Video RAM
VSNC	Vertical Synchronization
WAR	Write After Read
WAW	Write After Write
WB	Write Back

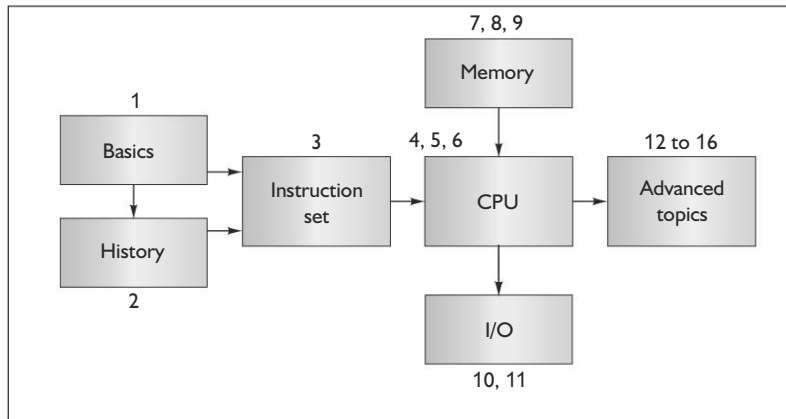
xxxvi *Abbreviation*

WE	Write Enable
WIW	Wide Issue Width
XOR	Exclusive OR
XR	Transceiver
Z	Zero

CHAPTER 1



BASIC COMPUTER ORGANIZATION



Chapter Flow Diagram

OBJECTIVES

- « *Stored program concept*
- « *Basic structure of computer*
- « *Functional units*
- « *Basic operational concepts*
- « *Hardware*
- « *Software*
- « *Operating system*
- « *Memory addressing*
- « *Memory operations*
- « *Bus structures*
- « *Computer types*
- « *Performance measurement*
- « *High performance techniques*

KEYWORDS

Memory Access Time	Compiler
Clock Frequency	Interpreter
Instruction Cycle	Machine Language
Memory Address Space	High Level Language
Word length	I/O Port
Random Access Memory	Hardware
Volatile Memory	Software
Booting	Microprocessor
Mainframe Computer	Architecture
Minicomputer	Organization
Microcomputer	Multiprogramming
Supercomputer	Cache Memory
Workstation	General purpose register
Personal Computer	Multiprocessing
Portable Computer	Pipelining
Application Software	SPEC Rating
System Software	Assembly Language
Operating System	



1.1 Introduction

Today computers are used in almost all walks of life: banks, hospitals, schools, shops, libraries, factories, courts, universities, prisons, etc. Computer buyers consider a number of factors, before choosing a product, such as good performance, low cost, easy programmability, low maintenance cost, less downtime, etc.

The design of a computer is highly sophisticated and complex. Over the years, a variety of concepts and techniques have been applied in designing and building it. The study of these concepts and techniques is the objective of this book.

This chapter serves as a foundation to the following chapters. It outlines the basic organization of a computer in order to prepare the reader for learning the various architectural and design issues covered in following chapters.



1.2 Man and Computing

The word *computing* means ‘an act of calculating’. The human race has so far seen three different types of computing:

1. Fully manual computing using brain and fingers.
2. Manual computing using simple tools such as slide rule, abacus, etc.
3. Automatic computing using a computer.

Table 1.1 compares the three types.

TABLE 1.1 Comparison of computing types

S. no.	Parameter	Manual computing	Manual computing with simple tools	Automatic computing
1	Speed	Low	Medium	High
2	Reliability	Poor; varies with individual	Medium	High
3	Problem complexity possible	Low	Medium	High
4	Extent of human effort	Very high	Medium	Very low
5	Consistency	Poor due to tiredness or moodiness	Medium	Very high
6	Impact	Some problems cannot be solved in reasonable time	Slightly better than manual computing	Any reasonable problem can be solved

The computer is a tool for solving problems (Fig. 1.1). Its basic ability is to perform arithmetic calculations. The computer is used to solve problems in several fields such as scientific, research, commerce, administration, manufacturing, etc. The following are some simple uses:

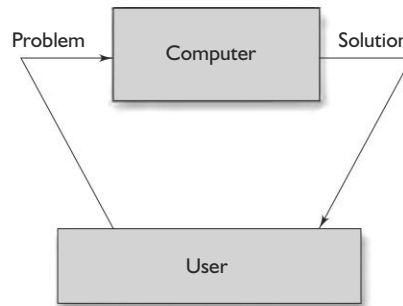


Fig. 1.1 User and computer

1. Calculating the average mark of students of a class.
2. Preparing monthly salary statement for a factory.
3. Designing a building.
4. Calculating the tax payable by a business firm.
5. Preparing materials list for production shop floor.
6. Preparing the results of an university examination.

The various advantages of using a computer are as follows:

1. Quick processing (calculations)
2. Large information storage
3. Relieving manual efforts
4. Messaging and communication

Combination of processing and storage has resulted in several new types of uses such as Multimedia, Internet, Desk Top Publishing (DTP), etc. in recent years.

1.2.1 Characteristics of a Computer

The main characteristics of a computer are listed below:

1. Very high speed of computation.
2. Consistency of behavior, unaffected by fatigue, boredom, likes and dislikes, etc.

3. Large storage capacity (for data and programs).
4. High accuracy of computation.
5. General purpose machine which can be programmed as per user's requirement.

1.3 Digital Computer

Most of the present day computers are digital computers though some are also analog computers. An analog computer monitors (senses) input signals whose values keep changing continuously. It generally deals with physical variables such as voltage, pressure, temperature, speed, etc. A digital computer operates on discrete (digital) information such as numbers. It uses binary number system in which there are only two digits 0 and 1. Each is called a bit (binary digit). Annexure 1 discusses the different number systems: decimal, binary, octal and hexa decimal. The digital computer is designed using digital circuits in which there are two different levels for an input or output signal. These two levels are known as logic 0 and 1. The modern computer is a digital computer capable of performing arithmetic (mathematical) and logical operations on data and generates a result (Fig. 1.2).

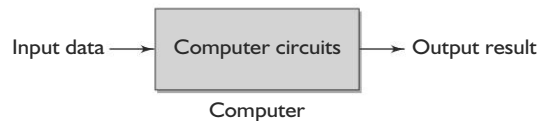


Fig. 1.2 Computer as an electronic machine

1.3.1 Program

The computer *behaves* as directed by the ‘program’ developed by the programmer.

To solve a problem, the user inputs a program to the computer (Fig. 1.3). The program has a sequence of instructions which specify the various operations to be done. Thus, the program gives a step-by-step method to solve a problem. A computer can carry out a

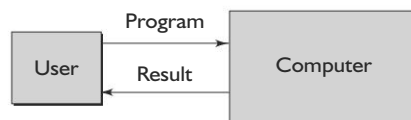


Fig. 1.3 Solving a problem

variety of instructions such as ADD, SUBTRACT, etc. The entire list of instructions for a computer is known as *instruction set*. A program uses combinations of the instructions according to the method (algorithm) to solve the problem. A program can be defined as a sequence of instructions with appropriate data for solving a problem (Fig. 1.4). The computer analyses each instruction and performs action on data.

1.3.1.1 Machine Language Program

The digital computer is an electronic machine. Hence, the computer program should consist of only 1's and 0's. The operations are specified in the instructions as binary information. The operands (data) are given as binary numbers. Such a program is called a machine language program. Though it is tedious to write programs in machine language, all early programs were written in machine language as there was no other choice. Different computer models had different machine languages and hence, a machine language program for one computer can not run on any other computer with different instruction set. Present day programmer uses high level languages due to simplicity of the high level languages. A high level language program consists of statements whereas a machine language program has instructions. The computer converts the statements into instructions.

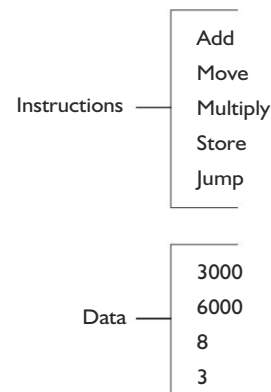


Fig. 1.4 A computer program

1.3.2 Hardware and Software

The term hardware generally refers to the electronic circuits in the computer. The main hardware modules are shown in Fig. 1.5. In practice, the term hardware is used for all physical items in a computer including mechanical, electrical and electronic assemblies and components. Figure 1.6 shows different hardware parts in a computer.

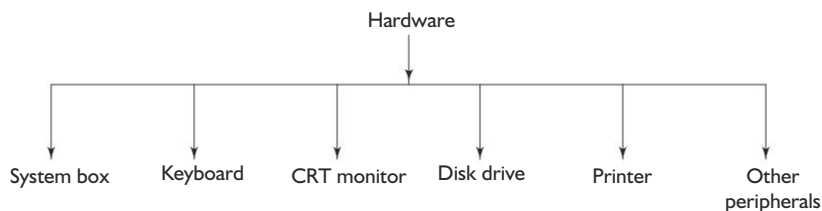


Fig. 1.5 Major hardware modules

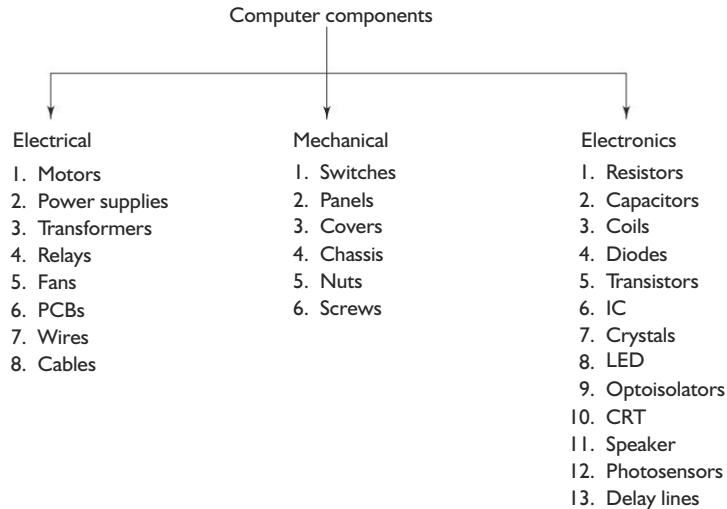


Fig. 1.6 Hardware components of a computer

Any program is a software. The software is developed to solve a problem and it controls the hardware when the program is executed. In other words, the hardware obeys the software as shown in Fig. 1.7. The hardware can be seen visually whereas the software is a logical action plan that is not visually noticeable. Here, when we say hardware, we don't mean the physical units but the functional units.

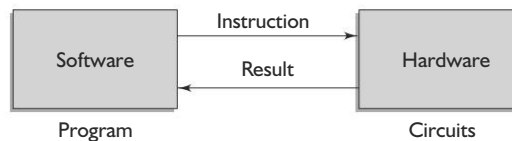
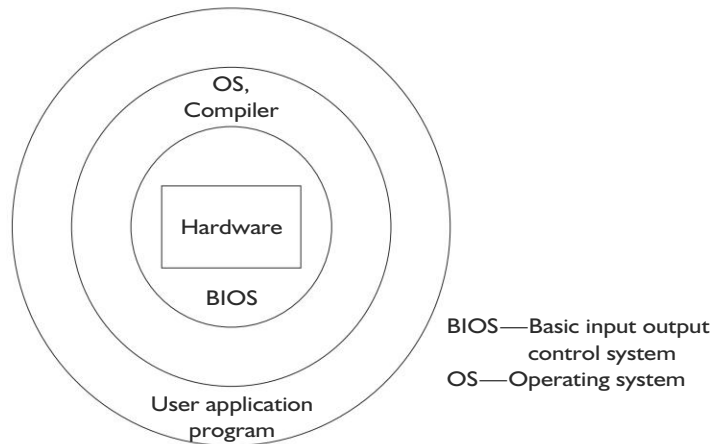


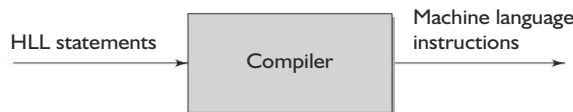
Fig. 1.7 Hardware–software interface

1.3.3 Layers in Modern Computer

A modern computer is not a mere electronic machine. It is a system consisting of multiple layers. The innermost layer is the hardware unit that is surrounded by other layers of software as shown in Fig. 1.8. A programmer writes an application program in a high level language using decimal numbers and English statements. The compiler is a language translator which converts the high level language program into equivalent machine language program, consisting of instructions and binary numbers (Fig. 1.9).

**Fig. 1.8** Layers in a computer

The operating system is a set of programs that provide a variety of functions with the objective of offering an efficient and friendly environment to the users and programmers. The following are important functions of operating system:

**Fig. 1.9** Use of compiler

1. Handling computer user requests for various services
2. Scheduling of programs
3. Managing I/O operations
4. Managing the hardware units

The Basic Input-Output control System (BIOS) is collection of *I/O drivers* (programs for performing various I/O operations) for different peripheral devices in the computer. These programs can be called by other programs whenever an I/O operation has to be done. The BIOS programs are requested by other programs to perform I/O operations whenever needed. Figure 1.10 illustrates the role of BIOS in a computer system. Chapter 10 discusses the three-tier interface between the device controller and the application program.

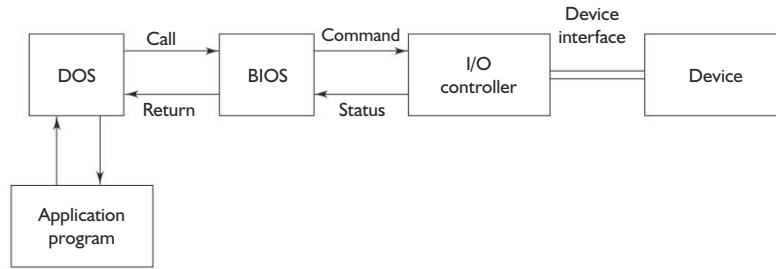


Fig. 1.10 Three-tier interface

1.3.4 Application Software and System Software

Computer software is classified into two types: application software and system software. An *application program* is a program for solving an users problem. Some typical examples are: payroll program, inventory control program, tax calculator, class room scheduler, library management software, train reservation software, billing software and game programs. Each of these programs is an application program since its objective is tackling a specific application (user requirement). A *system program* is a program which helps in efficient utilization of the system by other programs and the users. It is generally developed for a given type of computer and it is not concerned with specific application or user. The operating system and compiler are examples of system software. Annexure 2 provides additional information regarding various types of application software and system software.

1.4 Computer Organization and Functional Units

A modern computer is a computer system consisting of hardware and software. The hardware has five different types of functional units (Fig. 1.11): memory, arithmetic and logic unit (ALU), control unit, input unit and output unit. The program and data are

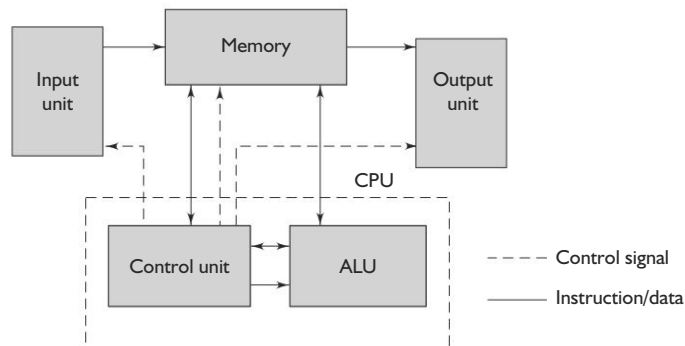


Fig. 1.11 Functional units in computer

10 Computer Architecture and Organization: Design Principles and Applications

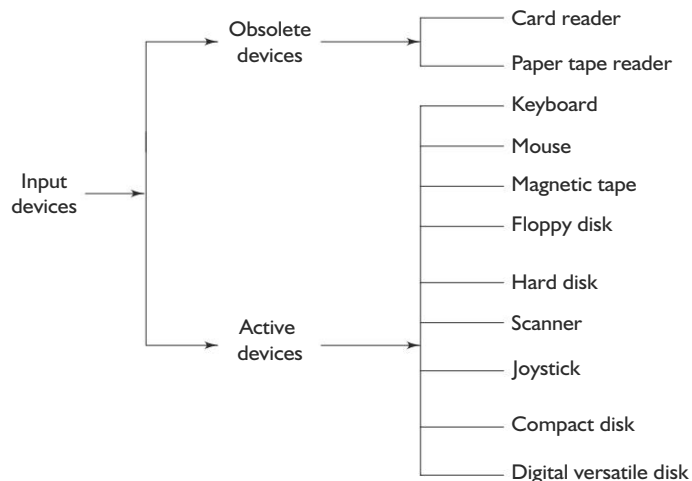
entered into the computer through the input unit. The memory stores the program and data. The control unit fetches and analyzes the instructions one-by-one and issues control signals to all other units to perform various operations. The ALU is capable of performing several arithmetic and logical operations. For a given instruction, the exact set of operations required is indicated by the control signals. The results of the instructions are stored in memory. They are taken out of the computer through the output unit. A computer can have several input units and output units. Figures 1.12(a) and (b) show examples of various input and output units. Table 1.2 gives brief summary of the functions of these devices. Some of them have dual functions: input and output.

TABLE 1.2 Peripheral devices and functions

S. No.	Peripheral device	Function/operation	Remarks
1	Keyboard	Detects the key pressed and sends the corresponding code	A common input device
2	Mouse	Coordinates of the mouse movement are sensed and sent to control the cursor; pressing a button enables selection of an item or action	Easy to operate pointing device; ideal for a graphical user interface (GUI)
3	Printer	Produces output on paper in readable form	Several types are available
4	Floppy disk drive	Magnetic recording of data on rotating floppy diskette by read/write head; two surfaces are present each handled by a separate head	A low cost secondary storage device; becoming obsolete due to popularity of CD and flash memory
5	Hard disk drive	Magnetic recording similar to floppy disk but flying heads record at higher density. Multiple disks of hard surface rotate at high speed	Better reliability and higher speed than floppy disk
6	Scanner	Converts pictures and text into a stream of data	Useful for publishing and multi-media applications
7	CRT Display (CRT monitor)	Displays characters and graphics	Internal operation has similarity with the picture tube in TV
8	Magnetic tape drive	Magnetic recording of data along the length of the tape	Common with older computers; nowadays used for data back-up
9	Plotter	Creates drawings on paper; graphics output device	A costly output device used for special applications such as CAD

(Contd.)

S. No.	Peripheral device	Function/operation	Remarks
10	Digital camera	Captures digital images and stores in its internal memory and transfers to computer	A modern popular input device
11	Compact disc (CD)	Provides large storage capacity (700 MB) using optical technology; laser beam is used for storing information	Slower than hard disk but faster than floppy disk; ideal for multi-media applications
12	Digital versatile disk (DVD)	New type of CD with storage capacity of 17 GB	Achieves theatre quality video and sound
13	Card reader	Senses holes on punched (paper) cards	Obsolete today
14	Card punch	Punches holes on (paper) cards	Obsolete today
15	Paper tape reader	Senses holes on paper tape	Obsolete today
16	Paper tape punch	Punches holes on paper tape	Obsolete today
17	Magnetic drum	Data is recorded on rotating magnetic drum	Slower than hard disk; obsolete today
18	Web camera	A low cost digital camera attachable to Internet	Helps posting photographs directly to web sites
19	Pen drive/RAM stick	A special kind of electrically erasable memory known as flash memory	Very small size; portable memory
20	Modem	Links a computer to a telephone line so as to communicate with a remote computer / device	Has become a household device thanks to the Internet


Fig. 1.12(a) Input units

The ALU and control unit have usually some temporary storage units known as registers. Each register can be considered as a fast memory with single location. Such registers temporarily store certain information such as instruction, data, address, etc. Storing in registers is advantageous since these can be read quickly compared to fetching them from external memory.

The ALU and control unit are together known as Central Processing Unit (CPU) or processor. The memory and CPU consist of electronic circuits and form the nucleus of the computer. The input and output units are electromechanical units consisting of both electronic circuits and mechanical assemblies. The input and output units are known as peripheral devices.

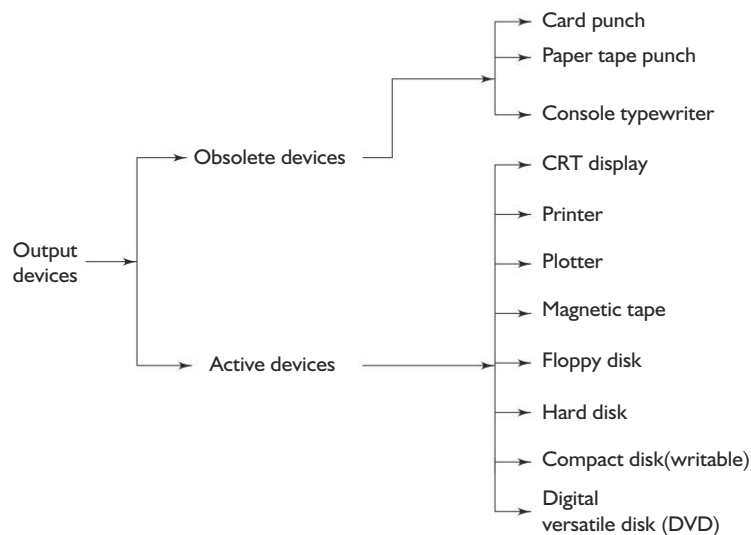


Fig. 1.12(b) Output devices

1.4.1 Stored Program Concept

All modern computers use the stored program concept which was initially conceived by the design team of ISA computer led by Von Neumann. Hence, it is commonly known as Von Neumann concept. The essentials of stored program concept are as follows:

1. The computer has five different types of units: memory, ALU, control unit, input unit and output unit.

2. The program and data are stored in a common memory.
3. Once a program is in memory, the computer can execute it automatically without manual intervention.
4. The control unit fetches and executes the instructions in sequence one by one. This sequential execution can be modified by certain type of instructions.
5. An instruction can modify the contents of any location in memory. Hence, a program can modify itself; instruction-execution sequence also can be modified.

1.4.2 Main Memory and Auxiliary Memory

The memory from which the CPU fetches the instructions is known as main memory or *primary memory*. Hence, to run a program, it has to be brought into the main memory. The *auxiliary memory* is external to system nucleus and it can store large amount of programs and data. The CPU does not fetch instructions of a program in the auxiliary memory. Several programs are stored in an auxiliary memory and the program that should be executed is brought into the main memory (Fig. 1.13). The auxiliary memory is cheaper compared to main memory and hence a computer generally has limited amount of main memory and large amount of auxiliary memory. The auxiliary memory is also known as *secondary storage*. Figure 1.14 lists several types of auxiliary memory. These are connected to the computer as Input-Output (I/O) devices.

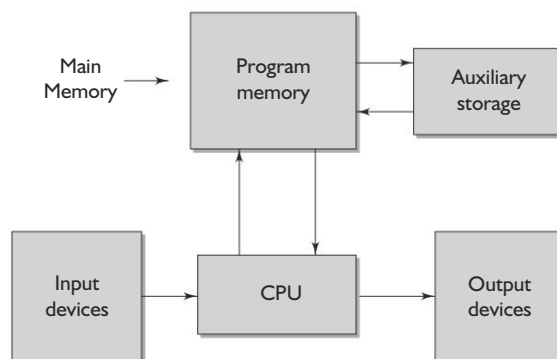
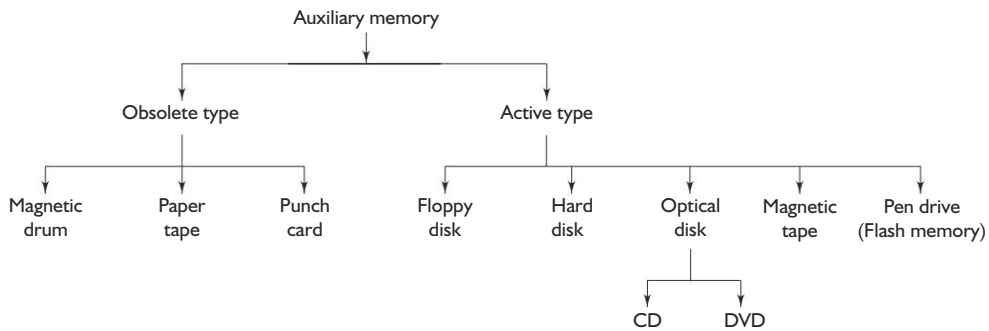
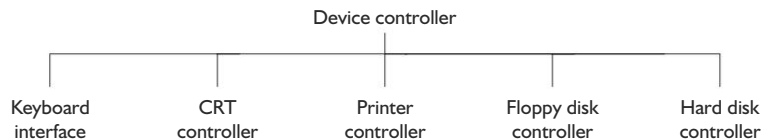


Fig. 1.13 Auxiliary storage

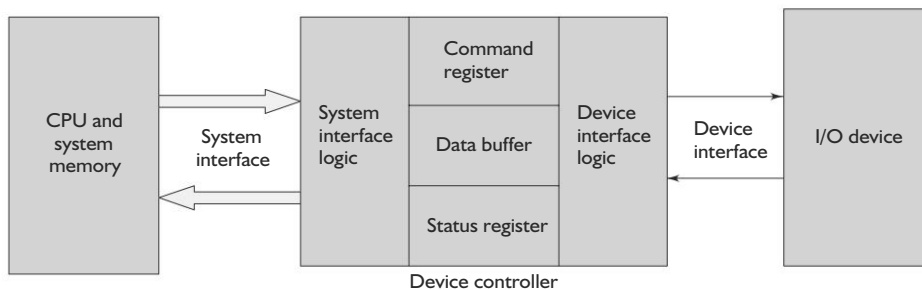
**Fig. 1.14** Auxiliary memory types

1.4.3 Device Controller

A peripheral device is linked to the system nucleus (CPU and memory) by a device controller, also known as I/O controller. Figure 1.15 shows different device controllers. The

**Fig. 1.15** Common device controllers

main function of a device controller is transfer of information (program and data) between the system nucleus and the device. Physically, a device controller can exist in three different forms: as a separate unit, as integrated with the device, as integrated with CPU. A basic device controller has five sections as shown in Fig. 1.16. It communicates with a device through the device interface that carries the signals between a device controller and a device. All device controllers communicate with CPU or memory through the system interface (Fig.1.17). The system interface is identical to all device controllers.

**Fig. 1.16** Device interface and system interface

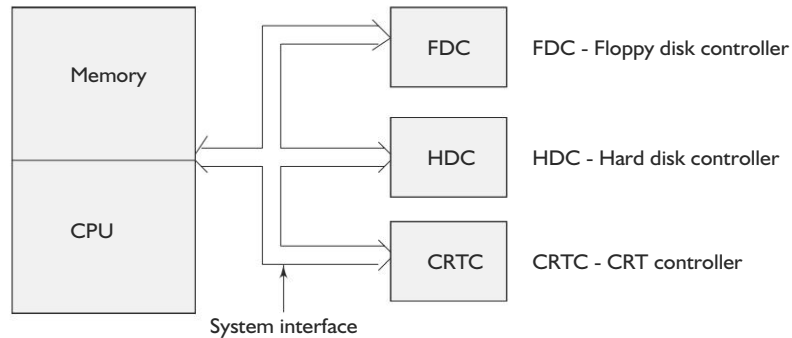


Fig. 1.17 Common system interface

The command register stores the command given by the software. The data buffer temporarily stores the data being transferred with the device. The status register stores the status of the device and controller. It is read by the software.

1.4.4 Device Interface Signals

There are three types of signals (Fig. 1.18) between a device and a device controller: data, control signals and status signals. The control signals are issued by the device controller

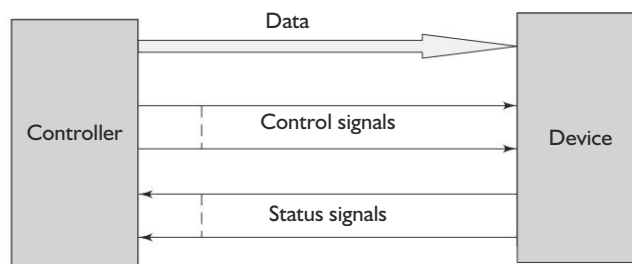


Fig. 1.18 Device interface signals

demanding certain actions by the device. For example, the RESET signal asks the device to get reset, i.e., clear the internal condition inside the device. The status signals are sent by the I/O device reporting certain internal conditions (status) to the device controller. For example, the ERROR signal reports that there is an error in the device. The data signals may be

sent either serially (Fig. 1.19), on one wire, bit-by-bit or in parallel (Fig. 1.20), on eight wires, carrying all eight bits of a byte simultaneously.

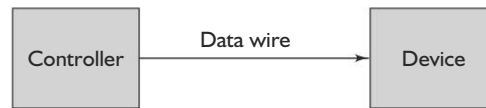


Fig. 1.19 Serial interface

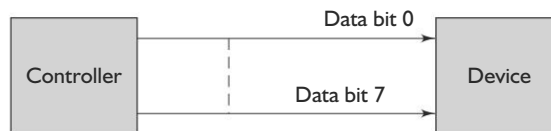


Fig. 1.20 Parallel interface

1.4.5 I/O Drivers

An I/O driver for a device consists of routines for performing various operations: read, write etc. Each routine gives appropriate commands to the I/O controller and the device controller issues necessary control signals to the device. To support a device in a system, three items are required:

1. A device controller which logically interfaces the device to the system nucleus.
2. A device interface cable which physically connects the device to the device controller.
3. An I/O driver.

An I/O driver (device driver) is a collection of I/O routines for various operations for a specific I/O device. It takes care of issuing commands to a device controller, verifying the status of the device controller and device and handling input/output operations. The operating system and other programs use the device driver routines for doing I/O operations. The I/O driver is a system program which can be asked by any other program to 'serve'. The I/O drivers of all the I/O devices are collectively known as BIOS.

1.5 Main Memory

The memory stores instructions, data and results of the current program being executed by the CPU. It is called *program memory* since the CPU fetches instructions only from this memory. The main memory is functionally organized as a number of locations. Each loca-

tion stores a fixed number of bits. The term *word length* of a memory indicates the number of bits in each location. The total capacity of a memory is the number of locations multiplied by the word length. Each location in memory is identified by a unique address as shown in Fig. 1.21. Two different memories with the same capacity may have different organization as illustrated by Fig. 1.22. Both have same capacity of 4 kilo bytes but they differ in their internal organizations.

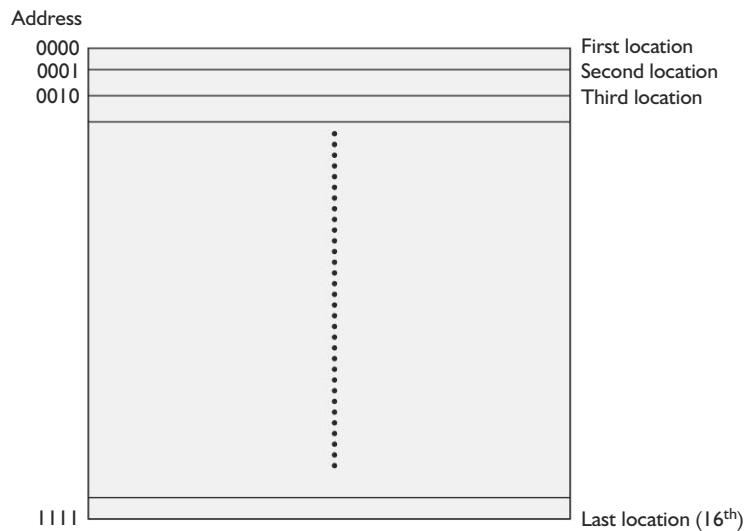


Fig. 1.21 Main memory locations

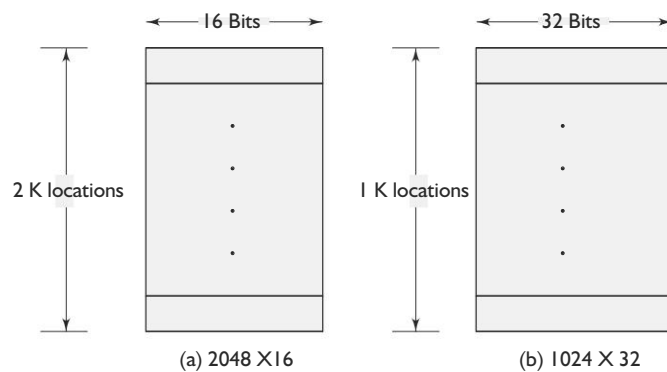


Fig. 1.22 Memory capacity and organization

Example 1.1 A computer has a main memory with 1024 locations of each 32-bits. Calculate the total memory capacity:

Word length = 32 bits = 4 bytes;

No. of locations = 1024 = 1 kilo = 1K;

Memory capacity = 1K $\times$ 4 bytes = 4 KB

1.5.1 Access Time and Cycle Time

The time taken by the memory to read the contents of a location (or write information into a location) is called its *access time*. Generally, main memory should have uniform access time for all its locations irrespective of its address (first, middle, last, etc.). Modern computers use semiconductor memory whose access time is around 50 ns. Old computers use magnetic core memory whose access time was around 1 μ s. After the access is over, the memory needs *settling time* after which the next access can be started. The settling time for semiconductor and core memory are in the order of 10 and 200 ns, respectively. The *cycle time* is the minimum time interval from the beginning of one access to the beginning of next access. It is the sum of access time and settling time.

1.5.2 Random Access Memory (RAM)

A memory which has equal access time for all its locations is called a Random Access Memory (RAM). Both semiconductor memory and magnetic core memory are RAM. Though core memory has become obsolete, its non-volatile nature is interesting. The semiconductor memory is volatile: its contents are lost when power supply is removed.

Example 1.2 A main memory has an access time of 45 ns. A 5 ns time gap is necessary from the completion of one access to beginning of next access. Calculate the bandwidth of the memory.

Access time = 45 ns; settling time = 5 ns;

Cycle time = access time + settling time = 45 ns + 5 ns = 50 ns;

Bandwidth = 1/cycle time = 1/50 ns = 20 MHz.

1.5.3 Read Only Memory (ROM)

In a modern computer, usually a small part of the main memory is a Read Only Memory (ROM). The CPU can read from the ROM but cannot write into it. Generally, some permanently required control programs and BIOS are stored in ROM by the manufacturer.

1.5.4 Memory Operations

The CPU addresses the memory both for a memory read operation and for a memory write operation. During a memory read operation, the CPU first sends the address of the location and then sends the memory read signal. On receiving the memory read signal, the memory starts reading from the location pointed by the address. After the access time, the content of the location is put by the memory on the data lines (Fig. 1.23). During a memory write operation, the CPU first sends the address of the location and then sends

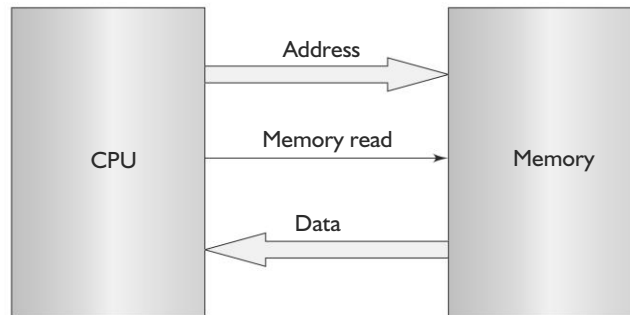


Fig. 1.23 Memory read operation

the data to be written and the memory write signal (Fig. 1.24). On receiving the memory write signal, the memory starts writing operation in the location corresponding to the address. Till the access time is over, the memory is busy with the write operation. The

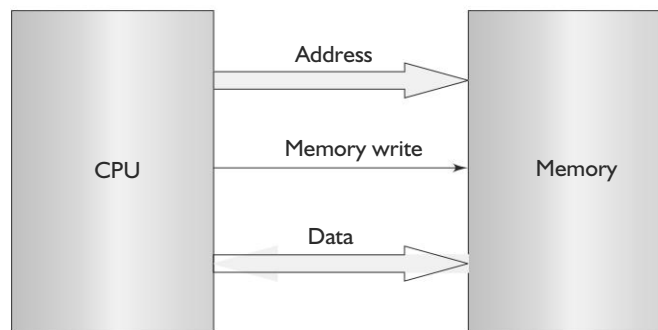


Fig. 1.24 Memory write operation

CPU uses two registers for communication with memory (Fig. 1.25). During read/write operations, the CPU puts the memory address in Memory Address Register (MAR). The Memory Buffer Register (MBR) is used to store the data from CPU during a write operation and the data from memory during a read operation. The MBR may be also called as Memory Data Register (MDR).

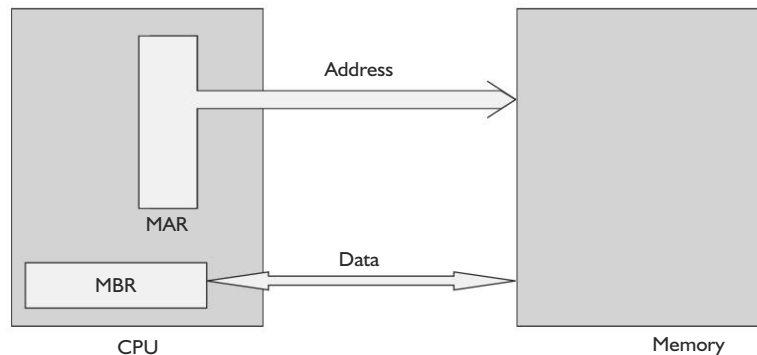


Fig. 1.25 CPU registers for memory access

1.5.5 Asynchronous and Synchronous Memory Interface

Two types of CPU–memory interfaces are implemented in different computer systems: synchronous and asynchronous. In synchronous interface (Fig. 1.26), the time taken by the memory for read/write operation is known to the CPU. Hence, there is no

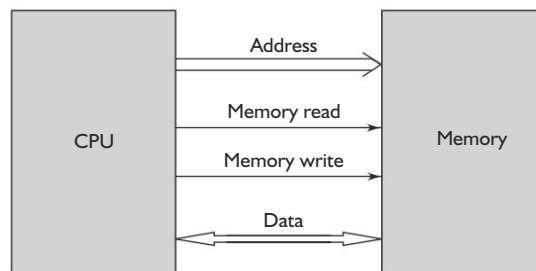


Fig. 1.26 Synchronous memory interface

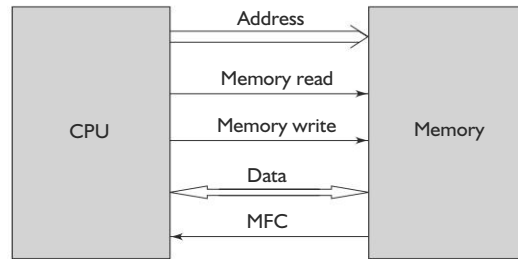


Fig. 1.27 Asynchronous memory interface

feedback from memory to CPU to indicate the completion of read/write operation. In asynchronous interface (Fig. 1.27), the memory informs about the completion of the read/write operation by sending a status signal, **Memory Function Complete** (MFC). This signal is also called as **Memory Operation Complete**.

1.5.6 Memory Addressability

The number of bits in the memory address determines the maximum number of memory addresses possible for the CPU. Suppose a CPU has n bits in the address, its memory can have a maximum of 2^n (2 to the power of n) locations. This is known as the CPU's memory addressability. It gives theoretical maximum memory capacity for a CPU. In practice, the physical memory size is less than this due to cost consideration.

Example 1.3 A CPU has a 12 bit address for memory addressing: (a) What is the memory addressability of the CPU ? (b) If the memory has a total capacity of 16 KB, what is the word length of the memory?

No. of address bits = 12;

Memory addressability = $2^{12} = 4$ kilo locations;

Memory capacity = 16 KB;

Word length = memory capacity/no. of locations = 16 KB/4K = 4B = 4 bytes.

Some popular CPU's and their memory addressability is given below:

CPU	No. of address bits	Memory addressability
IBM System 360/40	24	16 mega
Intel 8080	16	64 kilo
Intel 8088	20	1 mega
Pentium	32	4 giga
'Unknown'	40	1 tera



1.6 CPU Operational Concepts

The function of the CPU is executing the program stored in the memory. For doing this, the CPU fetches one instruction at a time, executes it and then takes up next instruction. This action is done repeatedly and is known as instruction cycle. As shown in Fig. 1.28, the instruction cycle consists of two phases: fetch phase and execute phase. In fetch phase, an instruction is fetched from memory. In execute phase, the instruction is analysed and relevant operations are performed.

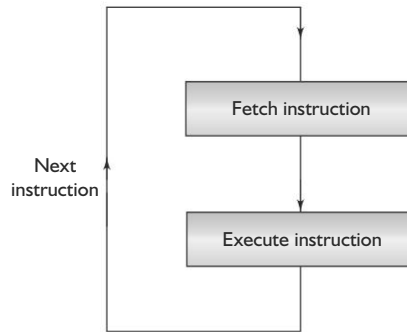


Fig. 1.28 Instruction cycle phases

1.6.1 Instruction Format and Instruction Cycle

The general format of an instruction is shown in Fig. 1.29. The operation code (opcode) field identifies the operation specified and the operand field indicates the data. Generally



Fig. 1.29 Instruction format

the operand field gives the address of the memory location where the operand (data) is stored. The complete sequence involved for the fetch and execute phases of one instruction is known as *instruction cycle*. Consider an ADD instruction whose format is shown in Fig. 1.30. The bit pattern in the opcode field identifies it as an ADD instruction. The



Fig. 1.30 ADD instruction format

other two fields identify the location where the two operands (data) are available. Figure 1.31 gives the steps involved in the instruction cycle and Table 1.3 defines each step. Figure 1.32 shows the formats of three more simple instructions: JUMP, NOOP and HALT.

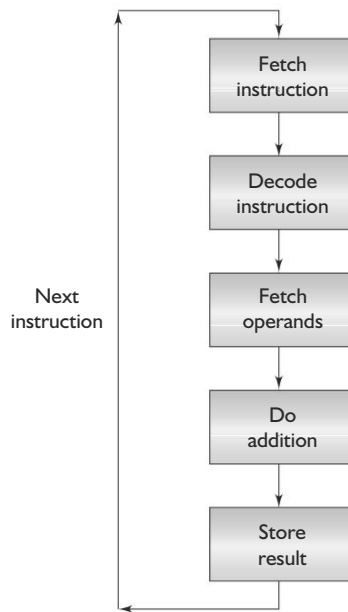
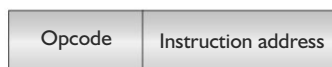


Fig. 1.31 Instruction cycle steps for ADD



(a) JUMP instruction



(b) HALT instruction



(c) NOOP instruction

Fig. 1.32 Formats of common instructions

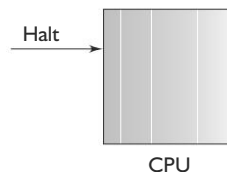
TABLE 1.3 Instruction cycle steps and actions for ADD

S. no.	Step	Action responsibility	Remarks
1	Instruction fetch	Control unit; external action	Fetches next instruction from main memory
2	Instruction decode	Control unit; internal action	Analyses opcode pattern in the instruction and identifies the exact operation specified
3	Operand fetch	Control unit; external (memory) or internal action depending on the location of operands	Fetches the operands, one by one, from main memory or CPU registers and supply them to ALU
4	Execute (ADD)	ALU; internal action	Specified arithmetic or logical operation is done
5	Result store	Control unit; external or internal action	Stores the result in memory or registers

1.6.2 CPU States

The CPU has two major operating states: *running* and *halt*. In running state, the CPU performs instruction cycles. In halt state, the CPU does not perform instruction cycle. When a computer is powered on, it goes to running state. The CPU goes from running state to halt state when any of the following actions take place:

1. *Software Halt*: A halt instruction is fetched and being executed by the CPU and hence the CPU halts.
2. *Halt Input*: The CPU receives halt input signal (Fig. 1.33) and halts. This signal may be generated either by the operator pressing the HALT (or PAUSE) switch in the front panel or by the circuits external to CPU.

**Fig. 1.33** Halt input signal

3. *Auto Shutdown*: During instruction cycle, the CPU encounters a serious abnormal situation and hence, does *shutdown* halting the instruction cycle. (This shutdown is different from system shutdown.)

1.6.3 CPU Registers

The CPU consists of following major registers (Fig. 1.34):

1. Accumulator (AC)
2. Program Counter (PC)

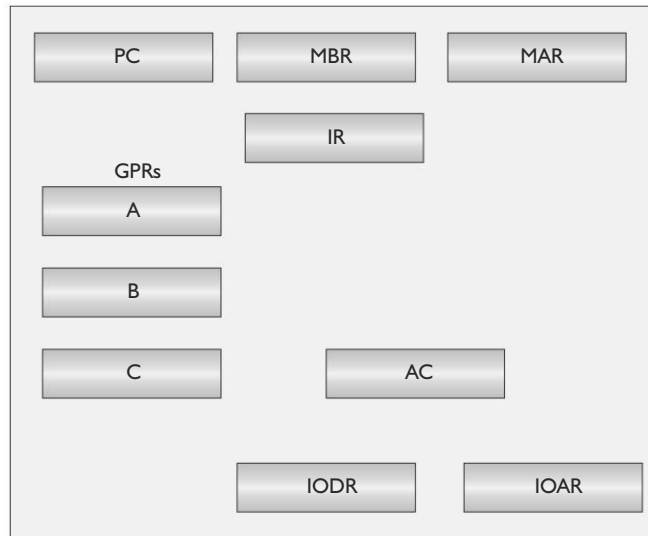


Fig. 1.34 CPU registers

3. Memory Address Register (MAR)
4. Memory Buffer Register (MBR)
5. Instruction Register (IR)
6. General Purpose Registers (GPR)
7. I/O Data Register (IODR)
8. I/O Address Register (IOAR)

The adder performs arithmetic operations such as addition, subtraction etc. The accumulator register holds the result of previous operation in ALU. It is also used as an input register to the adder. The program counter (instruction address counter) contains the address of the memory location from where next instruction has to be fetched. As soon as one

instruction fetch is complete, the contents of PC are incremented so as to point to the next instruction address. The instruction register stores the current instruction fetched from memory. The MAR contains the address of memory location during a memory read/write operation. The MBR contains the data read from memory (during read) or data to be written into memory (during write). The GPRs are used for multiple purposes: storing operands, addresses, constants etc. In addition to GPRs, the CPU also has some working registers known as scratch pad memory. These are used to keep the intermediate results within an instruction cycle for complex instructions such as MULTIPLY, DIVIDE etc.

1.6.4 CLOCK

The clock signal is used as a timing reference by the control unit. The clock signal is a *periodic waveform* since the waveform regularly repeats itself. The rate at which the periodic waveform repeats is known as the frequency (f). It is specified as cycles per second (cps) or Hz. The clock frequency is an indication of the internal operating speed of the processor. The fixed interval at which the periodic signal repeats is called as its time period (T). The relationship between the frequency and the period is

$$f = 1/T$$

The pulse width (t_{pw}) indicates the duration of the (clock) pulse. Another related term is duty cycle. The duty cycle is the ratio (expressed in percentage) of pulse width to period, i.e., duty cycle = $(t_{pw}/T) \times 100\%$. Fig. 1.35 illustrates the terms, period, frequency, and duty cycle.

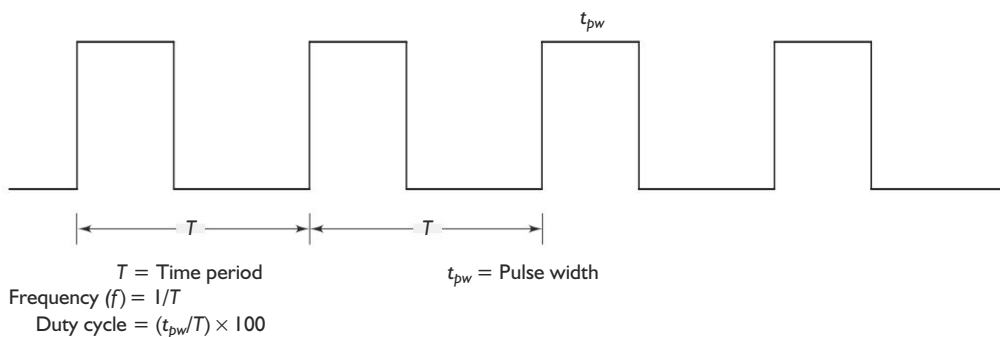


Fig. 1.35 Periodic waveform

Example 1.4 A clock signal has a frequency of 10 MHz with a duty cycle of 50%. Calculate its period and pulse width.

Data: $f = 10 \text{ MHz}$; duty cycle = 50%;

Time period, $(T) = 1/f = 1/(10 \times 10^6) = 100 \text{ ns}$;

Duty cycle = $(t_{pw}/T) \times 100 = 50\%$;

$t_{pw} = 0.5 \quad T = 50 \text{ ns}$.

1.6.5 Macro-operation and Micro-operation

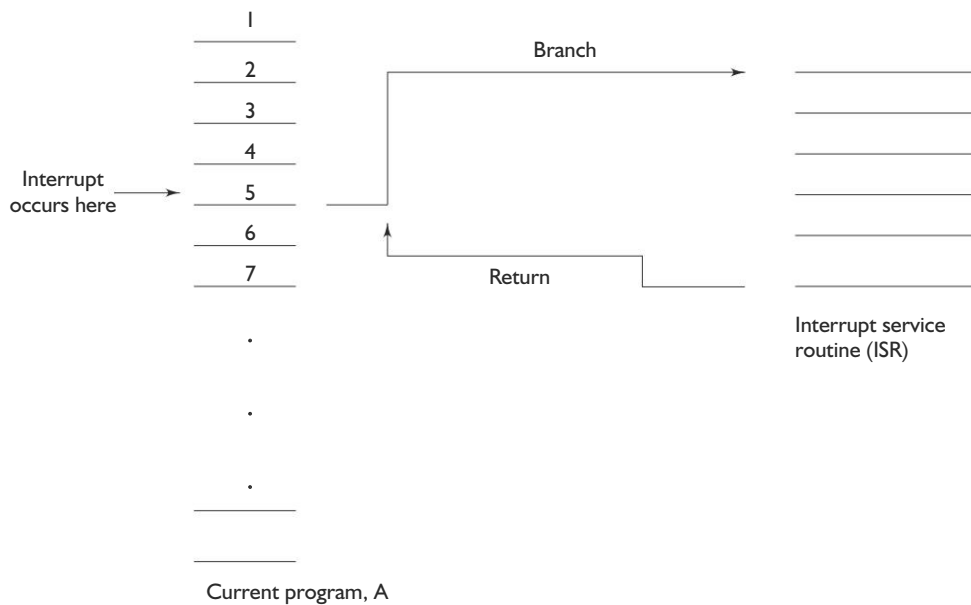
Every instruction specifies a macro-operation to be performed by the CPU. The opcode indicates the exact macro-operation. The instruction set lists all the possible macro-operations for a computer. A machine language program is also called macro-program. To execute an instruction, the CPU has to perform several micro-operations. The micro-operations are elementary operations inside the computer. The following are some examples of micro-operations: clearing a register, incrementing a counter, adding two inputs, transferring a register contents into another register, setting a flip-flop, complementing a register contents, shifting a register contents, reading from memory and writing into memory. Each micro-operation is done when the corresponding control signal is issued by the control unit. When a control signal is generated by the control unit, a *control point* in the hardware is activated resulting in the micro-operation. Table 1.4 gives some examples of micro-operations. Additional details of micro operations are discussed in Chapter 6.

TABLE 1.4 Sample microoperations

S. No.	Control signal	Micro-operation	Remarks
1	$MAR \leftarrow PC$	Contents of PC are copied (transferred) to MAR	The first micro-operation in instruction fetch
2	$PC \leftarrow PC + 1$	Contents of PC are incremented	The PC always points to next instruction address
3	$IR \leftarrow MBR$	Contents of MBR are copied to IR	The last micro-operation in instruction fetch
4	$MBR \leftarrow AC$	Contents of accumulator are copied to MBR	The first micro-operation in result store

1.7 Interrupt Concept

An interrupt is an event inside a computer system requiring some urgent action by the CPU. In response to the interrupt, the CPU suspends the current program execution and branches in to an Interrupt Service Routine (ISR). The ISR is a program that services the interrupt by taking appropriate actions. After the execution of ISR, the CPU returns back to the interrupted program. On returning from ISR, the CPU resumes the old interrupted program as if nothing has taken place. This means, the CPU should continue from the place (instruction address) when interruption has occurred and the CPU status at that time should be the same. For this purpose, the condition of the CPU should be saved before taking up ISR. Before returning from ISR, this saved status should be loaded back into the CPU. Figure 1.36 illustrates the interrupt concept. There are several types of interrupts



- Interrupt occurs after commencement of the 4th instruction. The CPU recognizes the interrupt before fetching the 5th instruction. The CPU suspends the current program execution and branches to ISR. After completion of ISR, the CPU returns to the interrupted program, A. Now the 5th instruction is fetched.

Fig. 1.36 Interrupt concept

TABLE 1.5 Common interrupt causes and actions

S. No.	Interrupt type	Cause for interrupt	Action by ISR
1	I/O completion	A device controller reports that it has completed an I/O operation i.e., a command execution is over	Control is given to I/O driver
2	Data transfer	A device controller reports that either it has a data byte ready (for input operation), or it needs a data byte (for output operation)	Control is given to I/O driver which performs data transfer
3	Overflow	The result of an operation is greater than the capacity of accumulator	Control is given to OS which may cancel the program or request user action

for various needs. Table 1.5 defines the causes for common interrupts. Chapter 10 covers interrupt handling in detail.

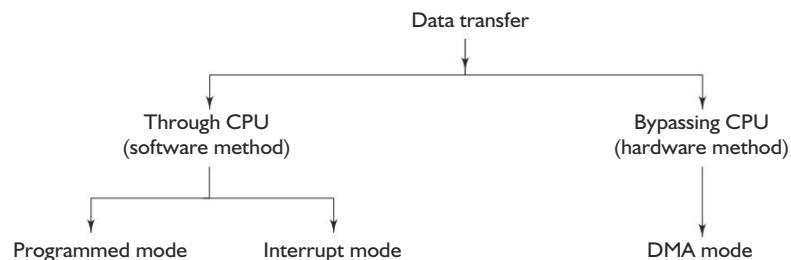
Nested interrupts: If the CPU is interrupted while executing an ISR, it is known as interrupt nesting. As a result, the CPU branches to new ISR. On completion of the new ISR, the CPU returns to the interrupted (old) ISR.

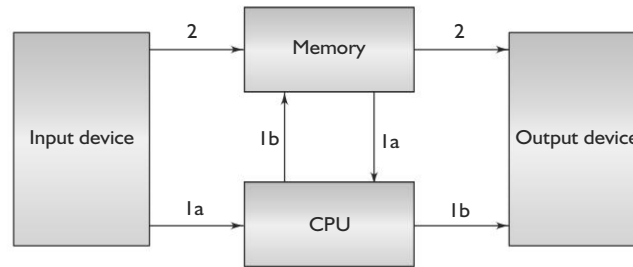


1.8 I/O Techniques

Input and output devices help us to give/take the program, data, and results to/from the computer. When an input operation is performed, we move information from an input device to memory (or CPU). Similarly, an output operation moves information from memory (or CPU) to an output device. An I/O routine (a program) takes care of input/output operations. It interacts with the device controller to perform information transfer (input/output).

An I/O routine can follow three different methods for performing data transfer as shown in Fig. 1.37 (a). In software methods, the I/O routine transfers every piece (byte) of data, through the CPU, in two steps as shown in Fig. 1.37 (b). An IN or OUT instruction is

**Fig. 1.37(a)** Methods of data transfer



1a, 1b—Two steps in software method; 2—hardware method

Fig. 1.37(b) Principles of data transfer

used for transfer of data between CPU and a device. Data transfers from high speed devices such as hard disk and floppy disk is difficult to handle by programmed mode or interrupt mode since transfer rate of these modes is slower than the rate at which data is coming from these devices. Hence, DMA mode is essential for high speed devices. Slow speed devices can be handled by programmed mode or interrupt mode. Interrupt mode is ideal for very slow speed devices so that CPU time is not wasted in waiting for device readiness between successive bytes transfer. Different I/O drivers follow different techniques to suit the device controller/devices. Chapter 10 covers the I/O techniques in detail.

1.8.1 I/O Ports

An *I/O port* is a program addressable unit which can either supply data to the CPU or which can accept data given by the CPU. An input port supplies data (when addressed) whereas an output port accepts data when addressed. An IN instruction is used to accept data from an input port (Fig. 1.38). An OUT instruction is used to supply data to an output port (Fig. 1.39).

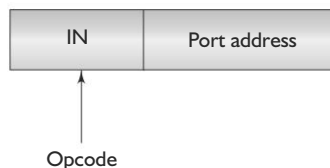


Fig. 1.38 In instruction

Each port has a unique address just like each memory location having a unique address. Physically, a port may exist in many forms: an independent unit, together with another port, or part of a special purpose component such as interrupt controller or device controller. Functionally, a port serves as a window for communication with CPU through data bus. For instance, the position (ON/OFF) of eight switches can be sensed through an eight bit input port. An IN instruction can transfer these status to a CPU register. Similarly, a set of eight LEDs can

be interfaced by an output port. An OUT instruction can supply the desired pattern by sending a CPU register contents to the output port. Chapter 10 covers I/O techniques in detail.

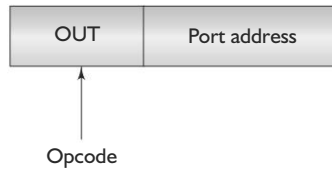


Fig. 1.39 Out instruction



1.9 Bus Concept

Connecting paths are necessary inside a computer for carrying following types of information between the subsystems (CPU, memory, and I/O controllers):

1. Instruction from memory to CPU.
2. Data from memory to CPU.
3. Data from CPU to memory.
4. Memory Address from CPU to memory.
5. Port Address from CPU to I/O controllers.
6. Command from CPU to I/O controllers.
7. Status from I/O controllers to CPU.

In mainframe computers, there is a separate path from each source register to each destination register (Fig. 1.40) for connecting the source and destination. Every source is directly

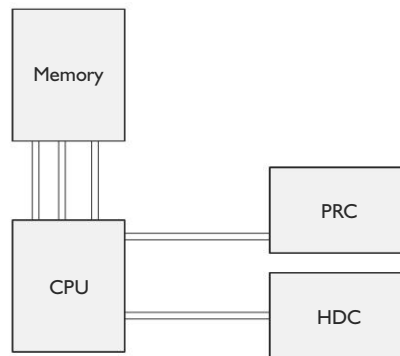


Fig. 1.40 Non-bus communication

connected to every destination. This results in huge cost since, there are lot of wires and driver/receiver circuits. In mini computers and microcomputers, bus concept is used for inter connection of signals between the subsystems in a computer. The bus is a shared common path for several sources and destinations (Fig. 1.41). A single wire carries a signal to multiple units. However, at any instant only two units are logically linked: one source and one destination. The other units are logically disabled though physically connected.

Suppose the CPU sends an eight bit data to floppy disk controller. The eight bits are sent on eight lines. Each line is common to all controllers and all of them receive the data pattern. Thus, each data bit is a bus and we have eight buses for data. In other words the width of the data bus is 8-bits. Though all controllers receive the data, only one controller is logically connected to the CPU. The other controllers do not use the data. The main advantage of bus is reducing cost of connecting paths (wires) and associated driver/receiver circuits. The disadvantage of bus is low speed since the bus is shared. At any time, only two units can communicate. Any other unit wanting to communicate has to wait.

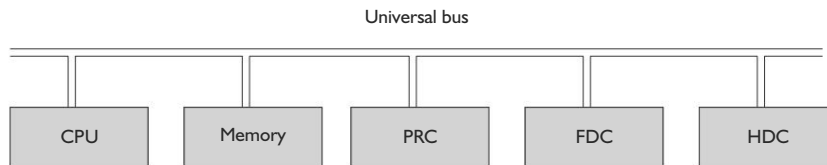
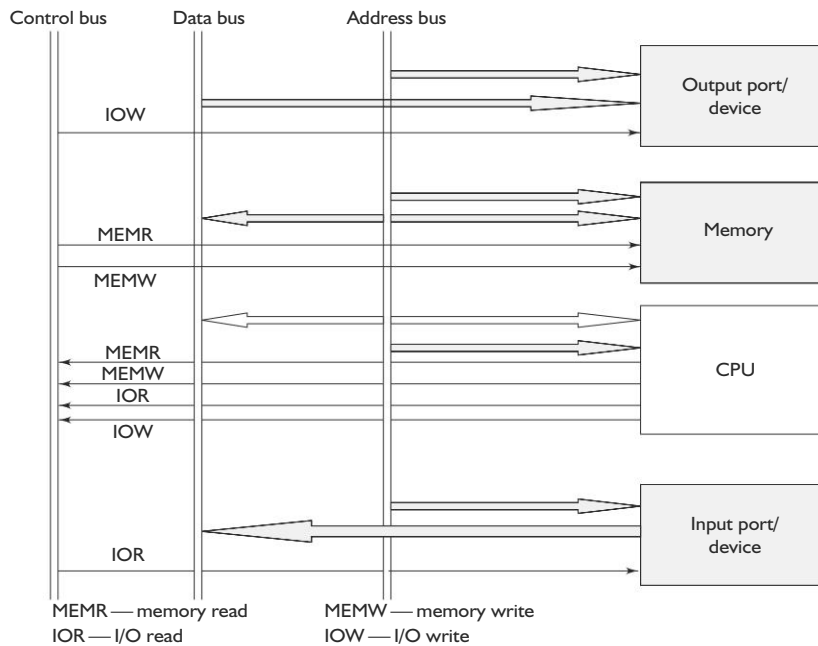


Fig. 1.41 Bus concept

Fig. 1.42 shows a simple bus structure. There are three different buses: data bus, address bus and control bus. The *data bus* is used for multiple purposes:



Not all control bus signals are shown.

Fig. 1.42 Simple bus structure

1. Communication between CPU and memory: instruction, operand and result.
2. Communication between CPU and I/O controllers: data, command and status.

The *address bus* is used by the CPU to send the address of the memory or I/O controller with whom it wants to communicate. The *control bus* has several control signals each

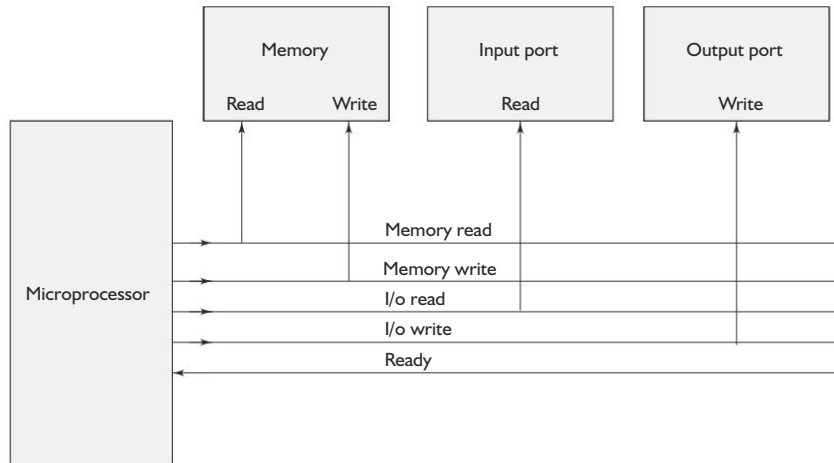


Fig. 1.43 Control bus signals

forming a bus. Some of them are: memory read, memory write, I/O read, I/O write, clock, reset and ready. Figure 1.43 shows the control bus signals and Table 1.6 defines them.

TABLE 1.6 Control bus signals definition

S. No.	Bus control signal	Signal Direction	Description
1	Memory read	Output of CPU	Indicates that the addressed memory should do memory read operation (and put data on data bus)
2	Memory write	Output of CPU	Indicates that the addressed memory should (take data from data bus and) do a memory write operation
3	I/O read	Output of CPU	Indicates that the selected input port should put data on data bus
4	I/O write	Output of CPU	Indicates that the selected output port should take data from data bus
5	Ready	Input of CPU	Indicates that the addressed subsystem (memory or I/O port) has completed the read/write operation

Microprocessors have two different bus structures (Fig. 1.44):

1. Internal bus which links the internal components of the microprocessor.
2. External bus which links the microprocessor with external units such as memory, I/O controllers, support chips etc.

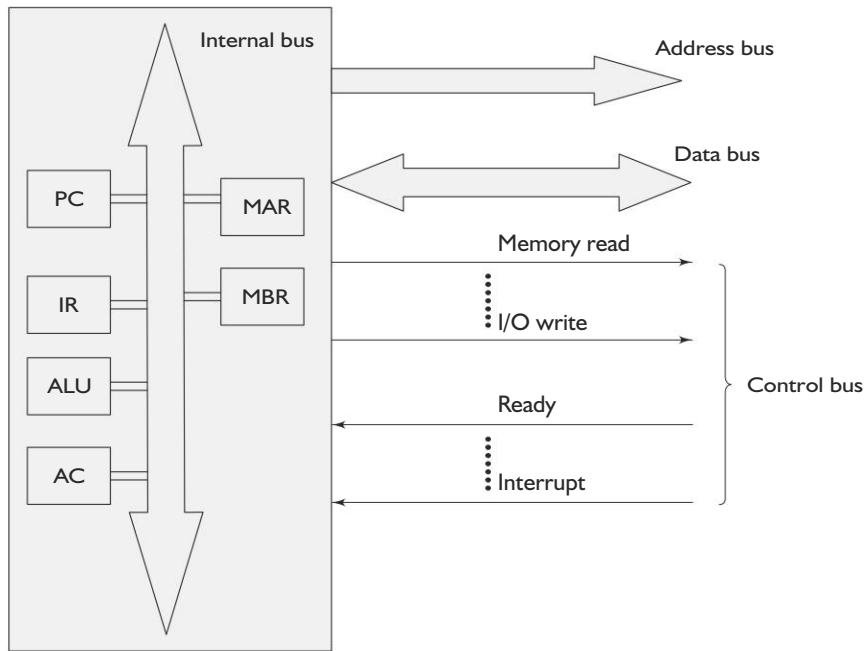


Fig. 1.44 Internal and external bus in microprocessor

1.9.1 Bus Cycle

In a bus based computer, communication between the CPU and other subsystems take place on the bus. A clearly defined protocol is followed on the bus so that there is no interference from other subsystems when the CPU is communicating with any one subsystem. Normally, the CPU can initiate a communication sequence whenever it wants. A sequence of events, performed on the bus, for transfer of one byte (or word), through the data bus, is called a bus cycle. There are four major types of bus cycles:

1. Memory read bus cycle: CPU reads from a memory location;
2. Memory write bus cycle: CPU writes data into a memory location;
3. I/O Read bus cycle: CPU reads (accepts) data from an input port;
4. I/O Write bus cycle: CPU writes (sends) data to an output port.

An instruction cycle may involve multiple bus cycles of different types. Fetching of an instruction involves memory read bus cycle. Storing a result in memory, needs memory write bus cycle. Fetching operands involve memory read bus cycles. An IN or OUT instruction execution involves I/O read bus cycle or I/O write bus cycle. Chapter 10 discusses the details of various bus cycles.

The CPU starts a bus cycle by sending the address (of memory or port) on the address bus. All the subsystems connected to the bus decode this address. Only the addressed subsystem will get logically connected to the bus and others do not interfere. The CPU indicates the type of bus cycle by sending the appropriate control signal (memory read, memory write, I/O read, I/O write) on the control bus. In a given bus cycle, the CPU sends one control signal which distinctly identifies the type of bus cycle. Memory responds to memory read or memory write whereas I/O ports respond to I/O read or I/O write. During output bus cycles, CPU puts data on the data bus and the addressed subsystem takes it. During input bus cycles, the addressed subsystem puts data on data bus and the CPU takes (reads) it.



1.10 System Software

A computer without system software is similar to a multi-storey building without lift. The main objective of any system software is simplifying system usage /operation and maximizing efficiency. A variety of system programs are available with different functions useful to various types of computer users: operators, programmers, maintenance engineers, etc.

1.10.1 Booting Sequence and OS

A computer system is always under the control of the operating system. Any other software gets control when the operating system assigns CPU to it. When a computer is installed, the operating system resides on hard disk. Immediately after powered on, the operating system should be loaded into memory and it should be given control of the CPU. The process of loading operating system (after powering on) into memory is called *booting*.

The following sequence of actions take place in the computer immediately after switching on:

1. The power-on action resets the CPU.
2. The CPU fetches its first instruction from a fixed location in main memory. This address is known as the *reset vector* for the processor. Usually this address belongs to ROM. In most computers, a Power – on Self – Test (POST) program starts at this address. The POST checks and verifies proper functioning of different hardware units.
3. The ROM contains a short program called *boot strap loader* which reads the boot record (track 0, sector 1) from the disk and stores it in main memory.

4. Then the boot strap loader gives control to the contents of *boot record* which is a program called boot program. The boot program loads the operating system from the disk to main memory in read/write area and gives control to the operating system.
5. The operating system takes over and issues message to the user.
Physically, the POST and boot strap loader programs are present in a ROM that also contains BIOS. Hence, this ROM is usually called as BIOS ROM.

1.10.2 Types of Operating Systems

In a simple computer system, the OS is small in size. In a powerful computer system, the OS consists of several program modules: kernel (supervisor), scheduler, process manager and file manager. The different types of OS are listed in Table 1.7. In a real computer system, the OS is hybrid one: a combination of different types.

TABLE 1.7 Operating system types

No.	Type	Remarks
1	Batch OS	Programs are input and run one by one
2	Inter-active OS	During program execution input of data is supported
3	Time-sharing OS	The system is shared by multiple users with terminals for interaction with OS
4	Multitasking/Multiprogramming OS	More than one task or program is in memory. The CPU keeps switching between programs.
5	Real-time OS	The OS periodically monitors the various inputs and accordingly different tasks are performed.
6	Multiprocessor OS	Runs many programs simultaneously on many CPUs in a single computer system.

1.11 Computer Types

Based on performance, size, cost and capacity, the digital computers are classified into four different types (Fig. 1.45): mainframe (or maxi) computer, minicomputer, microcomputer

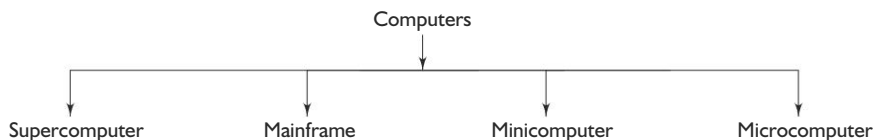


Fig. 1.45 Types of computers

and supercomputer. Due to constant change in technology and non-uniform standards among different computer manufacturers, any definition for these computer types is not easily accepted by everyone. Table 1.8 should be read with this caution in mind. A large system (mainframe) is physically distributed into more than one or two cabinets. It is an expensive computer that can be purchased only by large organizations. The minicomputers were developed with the objective of bringing out low cost computers so that smaller organizations could afford them. Some of the hardware features available in mainframes were not included in the minicomputer hardware in order to reduce the cost. Some features which were handled by hardware in mainframe computers were done by software in minicomputers. As a result, the performance of minicomputer is less than that of mainframe. The invention of microprocessor (single chip CPU) gave birth to the microcomputer that is several times cheaper than minicomputer but at the same time it offers everything available in minicomputer at a low speed. This has practically killed the minicomputer market.

TABLE 1.8 Classification of computers

Computer type	Specific computers	Typical feature	Typical word length
Mainframe	B 5000, IBM S/360	Large size, high cost	32 bits and above
Minicomputer	PDP-8, HP 2100 A, TDC-312	Medium size, low cost	8/16 bits
Microcomputer	IBM PC, TRS 80	Very small size, Very low cost	4/8/ 16/32 bits
Supercomputer	CRAY-1, Cyber 203/205	Extremely high speed	64 bits and above

Any computer that is designed with a microprocessor is a microcomputer. A wide range of microcomputer is available today. The microcomputers are classified in two ways:

1. Based on functional usage or application
2. Based on shape or size

Figure 1.46 illustrates the different models in both the classifications. Tables 1.9 and 1.10 define these models briefly. The portable computers operate on battery and are designed to consume low power. Special engineering and hardware design techniques are adopted to make the portable computers strong and light weight. Hence, these are costlier though smaller in size.

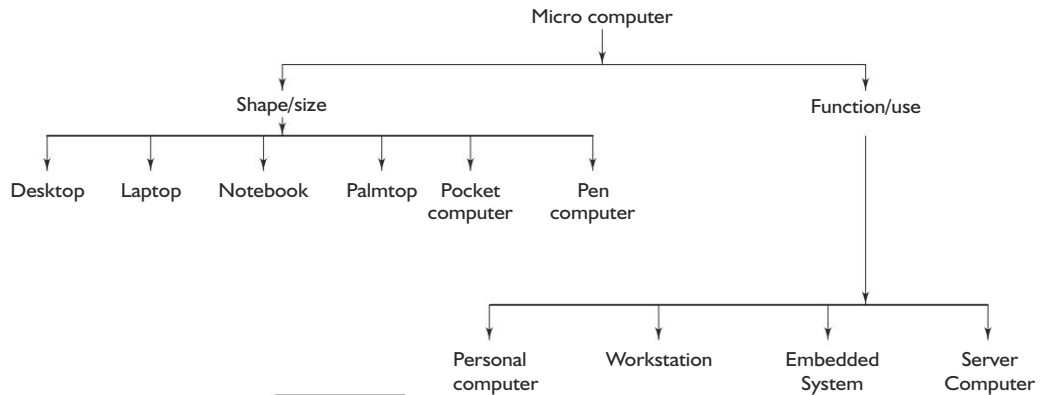


Fig. 1.46 Microcomputer classification

TABLE 1.9 Functional classification of microcomputers

S. No.	Model	Remarks
1	Personal Computer (PC)	A low cost general purpose microcomputer; used for common applications by various professionals and small and medium scale organizations; both initial price and ongoing maintenance cost is low.
2	Workstation	A special purpose microcomputer designed exclusively for a specific application such as CAD, DTP, Multimedia etc; provides high performance but costlier than PC.
3	Embedded System	A small dedicated/special computer (with control software) normally housed inside an equipment or device such as washing machine, digital set-top box, cell phone, automobile etc.
4	Server	Several computers (clients) are supported by a central server computer with powerful hardware and software resources common to all clients. Individual computers have limited essential resources such as CPU and memory. For the other required hardware/software, the server will provide service on request basis.

TABLE 1.10 Physical classification of microcomputers

S. No.	Model	Remarks
1	Desktop	Also known as tabletop; a common type
2	Laptop	Can be kept on lap of the user; a portable computer
3	Notebook	Appears like a notebook in shape; a portable computer
4	Palmtop	Fits on the palm of the user
5	Pocket	Like a pocket calculator
6	Pen	Looks like a pen



1.12 Computer Performance Factors

The performance of a computer is measured by the amount of time taken by the computer to execute a program. One may assume that the speed of the adder is a indicator of its performance, but it is not so. This is because the instruction cycle for ADD instruction involves not only add operation but also other operations such as instruction fetch, instruction decode, operand fetch, storing result, etc. The factors that contribute to the speed of these operations are as follows:

1. Instruction fetch: memory access time
2. Instruction decode: control unit speed
3. Operand address calculation: (1) GPRs access time/memory access time (2) address addition time
4. Operand fetch: memory access time/GPRs access time
Execute: addition time
5. Store result: main memory access time/GPRs access time

For two computers operating at the same clock rate, the execution time for ADD instruction can be different if they have different internal organization. Also for a given computer, the time taken for different instructions are not equal. For instance, MULTIPLY instruction will take more time than ADD instruction whereas LOAD instruction will take less time. Hence, the type of instructions and the number of instructions executed by the CPU, while running a program, decides the time taken by the computer for a program. The *execution time* for a program (T_p) is related to the clock speed and the actual program by the following equation:

$$T_p = \frac{N_{ie} \times CPI}{F}$$

where N_{ie} is the number of instructions executed (encountered by the CPU; not total instructions in the program), CPI, the average number of clock cycles needed for an instruction and F , the clock frequency. It should be noticed that N_{ie} is not equal to the total number of instructions in the program. Some of the instructions in the program may not be executed at all i.e. they may be skipped because of program logic. Similarly, some instructions may be executed several times because of loops.

From the above performance equation, it appears that to reduce program execution time T_p , the following approaches can be taken: reducing N_{ie} , reducing CPI and increasing F . Reducing N_{ie} involves having less instructions in the compiled program which is related to the compiler efficiency and the instruction set. Reducing CPI involves better CPU design to shorten instruction cycle time. Increasing F involves higher clock frequency which depends

on technology. However, while designing a computer system, these points have to be considered together since improving one parameter may affect other parameters. Some high performance techniques are outlined in Section 1.14.



1.13 System Performance Measurement

Knowing the performance level of a computer system is a common requirement for computer users (buyers, programmers, etc). Performance comparison of two or more computers is a frequent need for making a choice between several computers.

For a given computer, there are two simple measurements that give us an idea about its performance:

1. Response time or execution time:

This is the time taken by the computer, to execute a given program, from the start to the end of the program. The response time for a program is different for different computers.

2. Throughput:

This is the work done (total number of programs executed) by the computer during a given period of time.

When we compare two computers, A and B , we say A is faster than B if the response time for a program is less on A than on B . Similarly, we say ' A is n times faster than B ' if

$$\frac{\text{Execution time for B}}{\text{Execution time for A}} = n$$

$$\text{In other words, } n = \frac{\text{Performance of A}}{\text{Performance of B}}$$

1.13.1 MIPS

In the early days, the term MIPS (Millions of Instructions executed Per Second) was commonly used to indicate the speed of a computer. Since instruction cycle time is different for different instructions, the MIPS value will differ for different instruction mix in the program. Hence, the MIPS value of a computer gives only a rough idea about system performance.

1.13.2 Benchmark Programs

Benchmark programs are evaluation programs which are used to compare the performance of different computers. The time taken by a computer system to execute a benchmark program serves as a measure of its performance. This is used for decision making by buyers, developers and marketing teams. There are two important aspects to be considered when we use benchmark programs:

1. Different instruction types have different execution times.
2. Number of occurrence of an instruction type varies with different types of programs.

The ability of a benchmark program to give reasonably accurate information about the performance of a computer depends on the instruction mix used in the benchmark program. Since application programs of different types (scientific, commercial, etc.) have different instruction mix, it is very difficult to develop a benchmark program which will truly stimulate all real life programs. Hence, separate benchmark programs are developed with different instruction mix for each type of application: scientific, commercial, educational, etc. While no benchmark can fully represent overall system performance, the results of a group of carefully selected benchmarks can give valuable information about real performance.

1.13.3 Benchmark Suites

A *Benchmark suite* is a collection of different types of benchmark programs. Running a benchmark suite on a computer gives a reasonably good idea about the performance level of a computer. Since there are a variety of application programs in a benchmark suite, weaknesses of one or more programs are compensated by the strengths of certain other programs in the benchmark suite. Thus, a benchmark suite helps in determining relative performances of two or more computers. However, the extent of accuracy of the result depends on the nature of the individual programs in the benchmark suite.

1.13.4 SPEC Rating

The SPEC (System Performance Evaluation Corporation) is a non-profit organization dedicated for performance evaluation. It selects typical application programs for different application areas and publishes (announces) the performance results for important commercial computers. The performance of one of the commercially available computer is taken as the reference computer. The performance of other computers are rated as a relative measure of the standard computer. The following are some of the standards released by SPEC:

1. Desktop Benchmarks: SPEC89, SPEC2000, SPEC2006
2. Java Benchmarks: JSB2000, JVM98
3. Web server Benchmarks: SPECweb99, SPECweb96
4. Mail server Benchmark: MAIL2001
5. NFS Benchmark: SFS97

The SPEC rating for a computer X is specified as follows:

$$\text{SPEC rating for X} = \frac{\text{Program execution time for standard computer}}{\text{Program execution time for X}}$$

In practice, a suite of programs of various types is run and the SPEC rating is calculated for each. The SPEC rating is a reflection of multiple factors: CPU performance, Memory performance, System organization, Compiler efficiency, Operating system performance etc.

SPEC does two different roles: developing benchmarks and publishing results.

Developing suites of benchmarks: These suites have sets of benchmark programs with source codes and tools. The programs in the suites are developed by SPEC from programs donated by various sources (generally system manufacturers). SPEC aims on portability and creates tools and meaningful functions for the benchmarks programs.

Publishing news and benchmark results: Along with performance results, SPEC also provides additional information such as submitted results, benchmark descriptions, background information, and tools; these help in performance comparisons.

1.13.5 SPEC Groups

Presently SPEC has become an umbrella organization with three groups: OSG, HPG, and GPC.

Open Systems Group (OSG): OSG group develops benchmarks for desktop systems, workstations and multi-user servers supporting open operating system environments.

High Performance Group (HPG): HPG group develops benchmarks for symmetric multiprocessor systems, workstation clusters, distributed memory parallel systems, and traditional vector and vector parallel supercomputers. These benchmarks represent large, real applications, in scientific and technical computing.

Graphics Performance Characterization Group (GPC): Responsible for industry standard graphics benchmarks for graphical and multimedia applications, subsystems, OpenGL, etc.

Example 1.5 A benchmark program titled ‘quake’ (part of SPEC2000 suite) was run on two different computers: A with OPTERON processor and B with INTEL Itanium 2 processor. The computers A and B measured execution times of 72.6 seconds and 36.3 seconds respectively. The same program takes 1300 seconds on the Sun Ultra 5 – the reference computer of SPEC2000. Calculate the spec ratio of these two processors and comment on the relative performance.

$$\text{SPEC ratio of OPTERON system} = \frac{1300}{72.6} = 17.92$$

$$\text{SPEC ratio of Itanium 2 system} = \frac{1300}{36.3} = 35.78$$

Comment on relative performance: SPEC ratio of Itanium 2 is twice higher than SPEC ratio of OPTERON.

1.13.6 SPEC Suite and Mean

A SPEC suite has multiple application programs and each program gives a different SPEC ratio for a given computer, say X. For example, the SPEC 2000 suite has 14 benchmark programs. The mean of all the SPEC ratios of these programs gives a typical idea about the suite’s performance on computer X. Suppose $s_1, s_2, \dots, s_{14}$ are the SPEC ratios of different benchmark programs for X, then the geometric mean for the suite for X is calculated as

$$\text{Geometric mean} = n \sqrt[n]{\prod_{i=1}^n s_i}$$

where s_i is the SPEC ratio for program i.

Example 1.6 SPEC 2000 was run on Intel Itanium 2 and the following SPEC ratio figures were calculated for the 14 benchmark programs of SPEC 2000 as 28.53, 43.85, 27.36, 41.25, 12.99, 72.47, 123.67, 35.78, 21.86, 16.63, 18.76, 16.09, 15.99, and 11.27 respectively. Calculate the geometric mean.

Applying the formula, Geometric mean = 27.12



1.14 High Performance Techniques

Several techniques have been developed over the years to increase the system performance. A brief discussion of these is given here and exhaustive coverage of them is presented in later chapters.

1.14.1 High Performance Computer Architecture

The designer can follow any of the following two approaches for enhancing the performance of a processor:

1. Increasing the clock frequency.
2. Increasing the number of operations performed during each clock cycle.

Increasing the clock frequency depends upon IC fabrication process. For performing more operations per clock cycle, any of the two strategies are followed:

1. Providing multiple functional units in a single processor chip.
2. Executing multiple instructions concurrently by fully loading the functional units.

As discussed earlier in Section 1.12, the performance of a computer (i.e. the time taken to execute a program) depends on three factors:

1. Number of instructions to be executed
2. Average number of clock cycles required per instruction
3. Clock cycle time

As mentioned earlier, reducing the clock cycle time depends on the IC manufacturing technology. Research and newer inventions in this area continue. A convenient technique for obtaining high performance is parallelism. It is achieved by replicating basic units in the computer system. Parallelism has been successfully used since the early days in a variety of ways in several computer systems. Some common techniques followed by computer architects for achieving parallelism are as follows:

1. Using multiple ALUs in a single CPU.
2. Connecting multiple memory units to a single CPU to increase the bandwidth.
3. Connecting multiple processors to one memory for enhancing the number of instructions to be executed in a unit time.
4. Connecting multiple computers (as a cluster) that can share the program load.

1.14.1.1 Reducing Memory Access Time

If the memory access time is large, the instruction cycle takes more time. To resolve this, there are many techniques:

Cache Memory: Having a small and fast memory (between CPU and main memory) known as cache memory (Fig. 1.47) in which contents of certain main memory locations can be temporarily stored in advance. Whenever any of these locations are accessed, CPU need not access main memory since these will be supplied by cache memory very fast. Hence, CPU fetches an instruction or operand in shorter time than main memory access time. It is obvious that at any time the cache memory can keep only some location contents depend-

ing on its capacity. Generally cache memory is costlier as compared to main memory and hence its size should not be too big. Also the result of an instruction can be stored in cache memory instead of storing in main memory so that instruction cycle time for CPU is shortened.

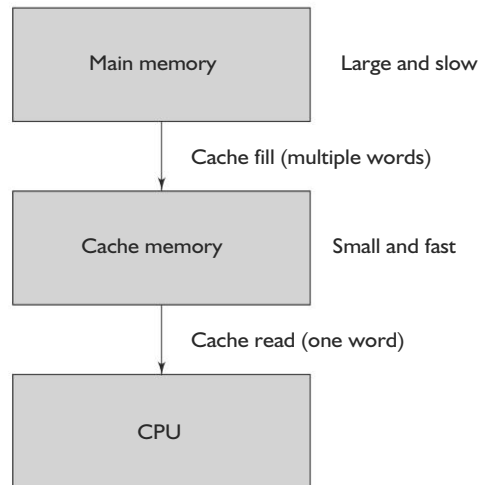


Fig. 1.47 Cache memory

Instruction Pre-fetch: Fetching an instruction in advance when previous instruction execution is not yet completed is known as instruction pre-fetch (Fig. 1.48). In this technique, two consecutive instruction cycles are overlapped. As a result, the instruction fetch time in the instruction cycle is reduced to zero.

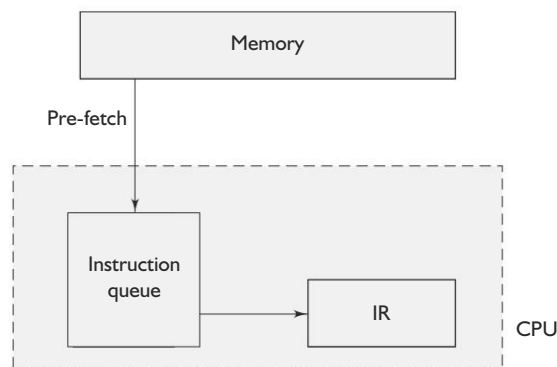


Fig. 1.48 Instruction prefetch

Memory Interleaving: Organizing the main memory into two independent banks with one containing odd addresses and the other containing even addresses is known as memory interleaving (Fig. 1.49). When CPU is accessing more than one consecutive location, both odd and even banks can perform simultaneous read/write operation. This effectively cuts the access time by half.

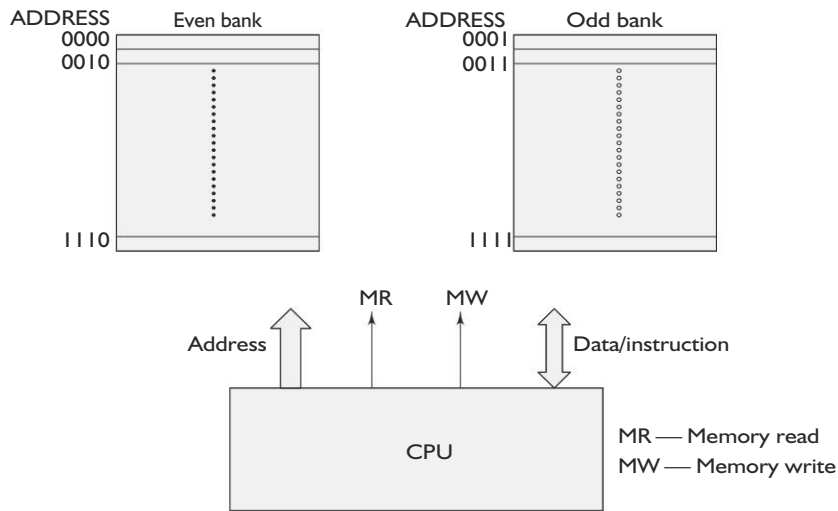


Fig. 1.49 Memory interleaving

1.14.1.2 Reducing Instruction Decode time

The time taken for decoding depends on hardware circuit technology. It can be overlapped with the execution time of previous instruction if it has been already fetched. This technique is known as *pre-decode* or instruction look-ahead (Fig. 1.50).

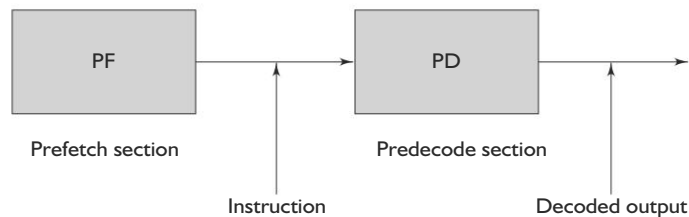


Fig. 1.50 Predecode

1.14.1.3 Instruction Pipelining

Even a fast adder needs some time to produce result. However, the CPU can be divided into different sections so that each stage of an instruction cycle can be done by an independent section. All sections can work simultaneously for different instructions. This technique is known as *instruction pipelining* (Fig. 1.51). The net effect of this full overlap of instruction cycle is that average instruction cycle time is equal to the time taken by the longest stage. Chapter 12 covers elaborately the technique of instruction pipelining.

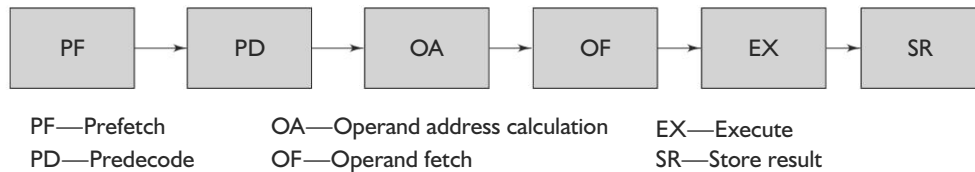


Fig. 1.51 Instruction pipelining

1.14.1.4 Superscalar Architecture

If more than one adder is available in a CPU, all adders can work simultaneously for consecutive instructions. This technique is known as *superscalar architecture* (Fig. 1.52). Generally two or more pipelines are present in a single CPU. Chapter 12 discusses the superscalar architecture.

1.14.2 Instruction set: CISC and RISC

There are two popular architectural standards for CPU design:

Complex Instruction Set Computing (CISC)

Reduced Instruction Set Computing (RISC)

The CISC is an old concept where as the RISC is a modern concept. A CISC CPU has a large number of instructions of which many are powerful. Its control unit is highly complex. A RISC CPU has a small number of instructions all of which are simple. Its control unit is very simple. All older systems are CISC systems but today's systems comprise both types. RISC systems are popular today due to their higher performance but they are costlier. Hence, they are used only for special applications where a special need for high performance or reliability arises. Chapter 3 discusses the CISC and RISC concepts briefly and Chapter 13 reviews the RISC systems in detail.

1.14.3 Compiler Efficiency

A good compiler generates a short machine language program (object code) for a given high level language program (source code) with minimum number of instructions. To what extent the instruction count can be small depends also on the CPU architecture. However the CPI also should be small so that the program execution time is small. This again depends on the CPU architecture. An optimizing compiler for certain advanced processors is a special compiler that uses several techniques to minimize the program execution time. Chapters 13 and 14 discuss about the special compilers.

The overall performance of a computer system can be improved by certain advanced system designs discussed in the following chapters. Two popular common design techniques are multiprogramming and multiprocessing which are outlined in Table 1.11.

Some of the high performance techniques are visible to operating system whereas others are invisible. The invisible design techniques are not part of system architecture but part of system organization.

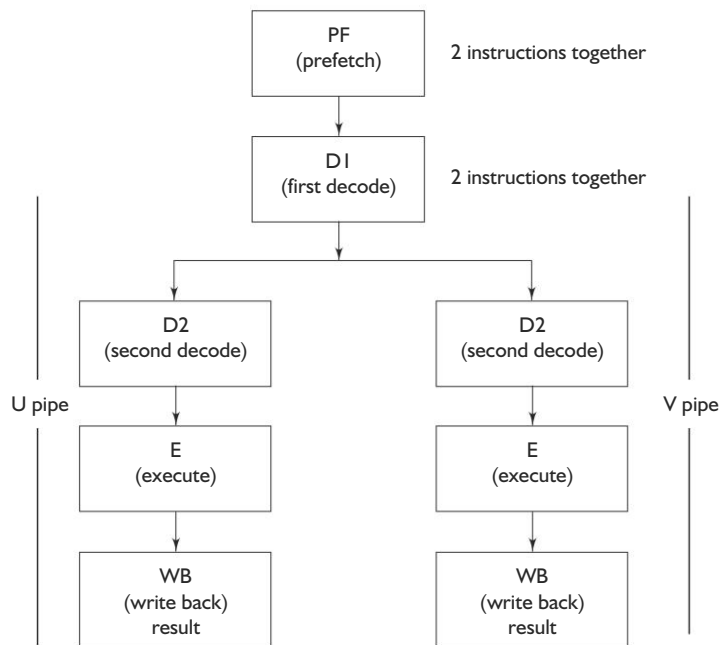


Fig. 1.52 Superscalar architecture (Pentium)

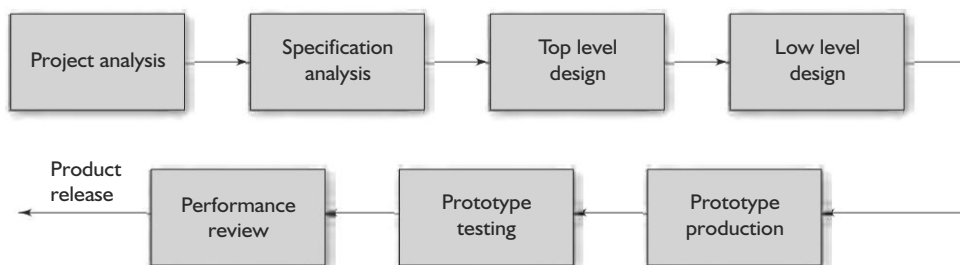
TABLE 1.11 Advanced system designs

S. No.	Design feature	Mechanism	Remarks
1	Multiprogramming	A single CPU multiplexes among multiple programs executing them concurrently; all programs stay in a common memory	CPU idle time is eliminated thereby improving system throughput
2	Multiprocessing	More than one CPU in a single system, with each CPU executing a different program; Each CPU may have a separate memory, or all CPUs may share a common memory	The OS is highly sophisticated



1.15 Computer Design Process

The development of a computer is a multi-step process, as shown in Fig. 1.53. The first step is project analysis. Here, the technical and commercial feasibility of the proposed product (computer) is studied and a feasibility report is prepared. In the next step, specifications analysis is done; using the feasibility report, the product specification is designed. For designing a computer, the following points are considered:

**Fig. 1.53** Computer development process

1. Expected performance range
2. Target application segment (business/scientific/server/client, etc.)
3. Desired price range
4. Essential features required so as to have edge over the competition.

These goals are finalised by the top management of the computer development company after discussions with marketing, development and engineering teams.

In the top level (system) design stage, the functioning modules and their specifications are finalised whereas, in the low level (detailed design) step, the design of individual functional modules are prepared. The prototype manufacturing stage deals with building and testing of the actual prototype. The results of this testing are reviewed and necessary changes are done to the design the documents. The prototype product is again revised and tested before handing it over to the manufacturing (production) teams.

Some of the issues involved in designing a new computer are as follows:

1. Is the new computer required to be 'software compatible' with any existing computer so that all presently used programs can be used with the new computer also?
2. If the new computer has a totally new architecture with a new operating system (to be developed), is it necessary to develop a 'simulator' (for the new computer) which runs on the existing computer so that programs developed for the new computer can be tested even before the new computer is ready?
3. Can any existing computer be used to 'emulate' the new computer to study the expected performance of the new computer?
4. Is it possible to 'import' any subsystem design from the existing computer to the new computer so that hardware development will be fast?

A given computer can be fully understood by studying its three dimensions: structure, function and performance.



1.16 Computer Structure

A computer's structure indicates its internal structure. The two aspects of a computer structure are: functional and physical. The functional structure identifies functional blocks and the relationships between them. The functional structure is an important factor for designing the computer as it reflects its performance. The physical structure identifies the physical modules and the connections between them. The physical structure do not affect the performance but it influences reliability and cost. Figure 1.54 gives a partial functional structure of a PC showing four subsystems: CPU, memory, CRT subsystem and keyboard subsystem. The system bus links all the subsystems.

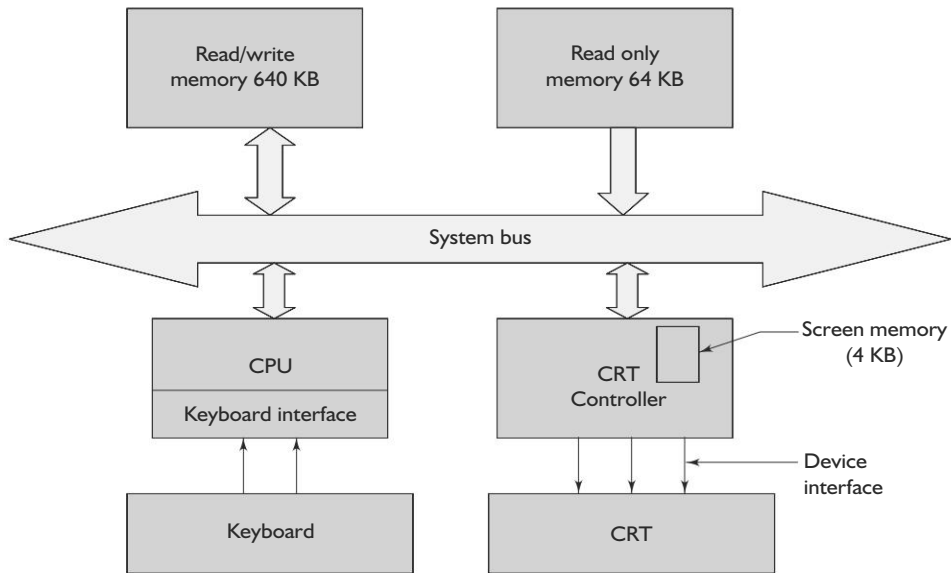


Fig. 1.54 Functional structure (partial) of a PC

Figure 1.55 gives the partial physical structure for the PC. The system board (CPU board) serves as a motherboard with I/O slots interconnecting the various multiple daughterboard

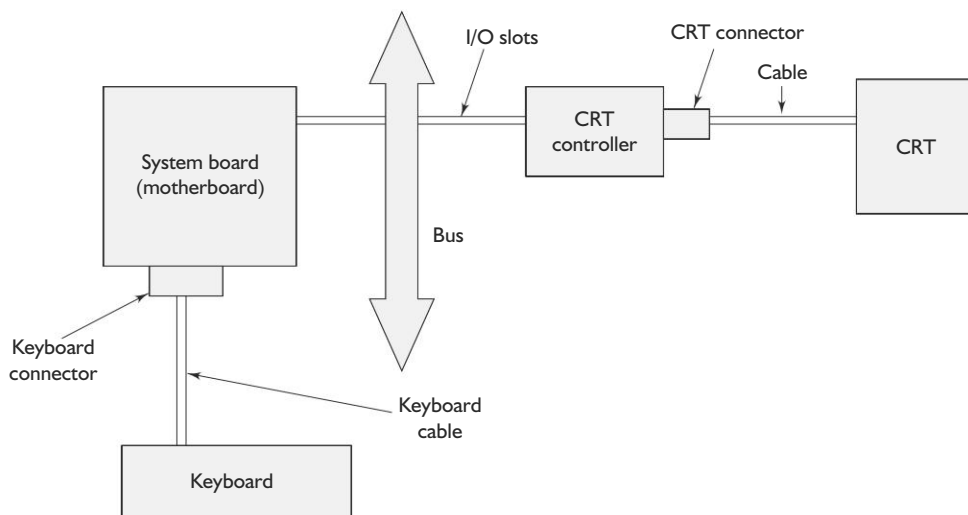


Fig. 1.55 Physical structure (partial) of a PC

PCBs. Generally, the daughterboard PCBs are controller PCBs or memory PCBs. The CRT controller is the only daughter board in this figure. The motherboard has four functional blocks: CPU, RAM, ROM and keyboard controller. The CRT controller has an on-board video buffer memory (screen memory).

The memory space has three parts:

1. Read/write memory (RAM) for user program and operating system.
2. ROM for BIOS.
3. Screen memory (video buffer). It is logically a part of the CPU's main memory but can be read by the CRT controller also.

For the same functional structure (Fig. 1.54) an alternate physical structure is shown in Fig. 1.56. The functional structure focus on its design whereas the physical structure shows the implementation or the layout.

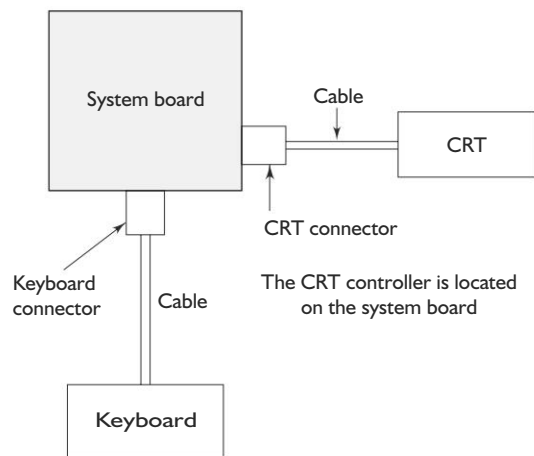


Fig. 1.56 Alternate physical structure for PC

Both functional structure and physical structure have a hierarchical composition, as shown in Figs. 1.57 and 1.58. There are various levels in a computer. The structure at a level does not indicate its behavior. Comparison of the physical structures of any two computers would reveal their components and technology. Similarly, the functional structure indicates its design but it does not indicate timing constraints or signal sequences.

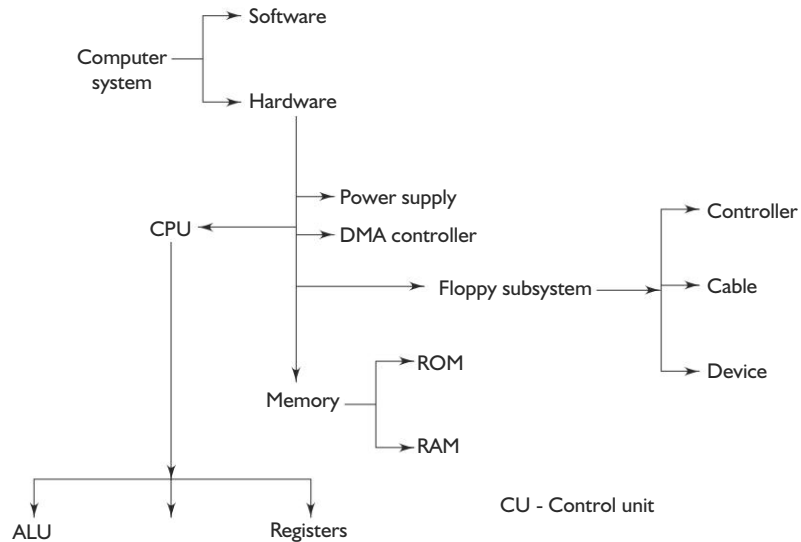


Fig. 1.57 Hierarchical view of functional structure of a system

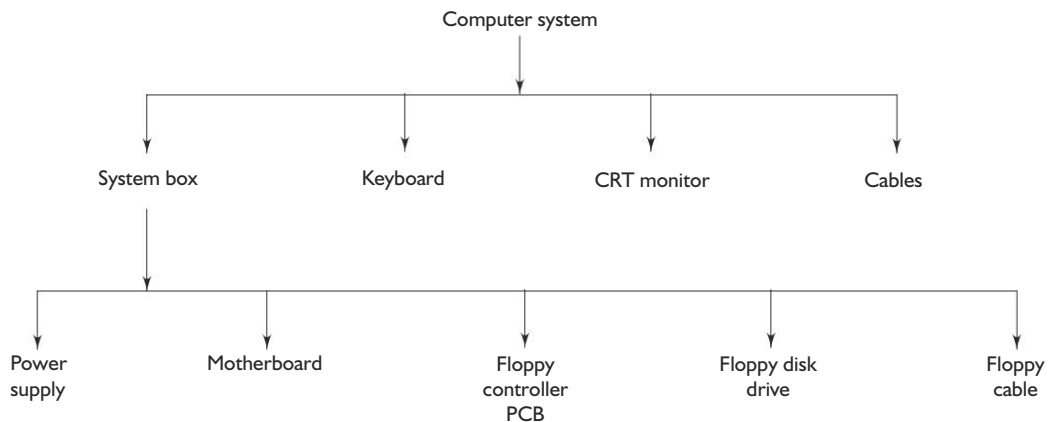


Fig. 1.58 Hierarchical view of physical structure of a PC

1.17 Computer Function

The functioning of the computer indicates its behavior. The output of a computer is a function of the input. At an overall level (i.e., as a system) its function is program execution.

It involves data storage, data processing and data transfer. Figure 1.59 illustrates the various levels and Table 1.12 identifies the functions at these levels.

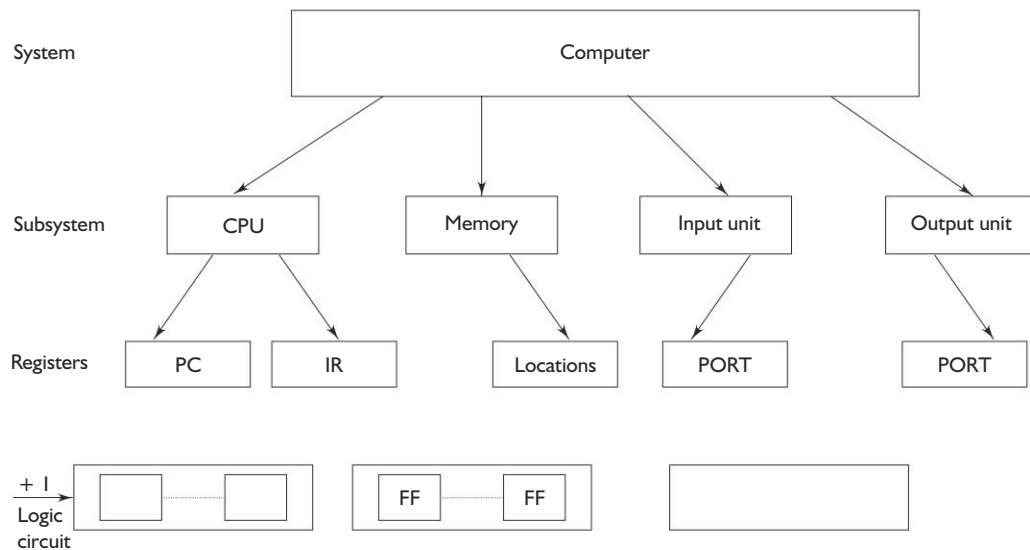


Fig. 1.59 Computer levels and functions

TABLE 1.12 Computer functions and levels

S. No.	Levels	Functions (sample cases)
1	System level	Program execution
2	Subsystem level	CPU: instruction processing; memory: storage; input unit: transferring user input; output unit: transferring system output to user
3	Registers level	PC points to next instruction address; AC contains current ALU operation result; IR stores instruction fetched from memory; MAR gives address of memory location for read/write operation
4	Logic circuit level	IR is made up of multiple D flip-flops with common clock input. The control signal $IR \leftarrow MBR$ triggers the register
5	Component level	Keyboard uses capacitor switches; when a key is pressed, the variation in capacitance between two plates is sensed as a key press



1.18 Architecture and Organization

A computer's architecture provides various attributes of the computer system which are needed by a machine language programmer or a system software designer to develop a program. It is a conceptual model providing the following information:

1. Instruction set
2. Instruction format
3. Operation codes
4. Operand types
5. Operand addressing modes
6. Registers
7. Main memory space utilization (memory map)
8. I/O space allocation (I/O map)
9. Interrupts assignment and priority
10. DMA channels assignment and priority
11. I/O techniques used for various devices
12. I/O controller command formats
13. I/O controller status formats

Figure 1.60 shows the computer architectural aspects needed for a machine language programmer. Computer organization gives an in-depth picture of its functional structure and logical

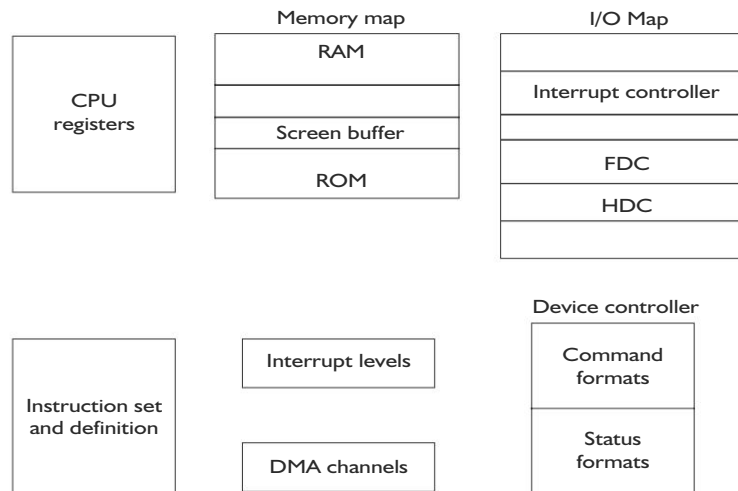


Fig. 1.60 Computer architecture for a programmer

interconnections between the different units (functional blocks). Usually it includes the details of the hardware. Two computers with same architecture can have different organization. Similarly, two computers having same organization may differ in their architecture. While designing a computer, its architecture is fixed first and then its organization is decided.

1.18.1 Implementation

The implementation of a computer's architecture (and organization) takes the form of hardware circuits grouped into functional blocks. Physical implementation of the computer design depends on its technology. Two computers of same organization designed at two different periods would have different component technology. Further, the architecture of these two may or may not be same.

SUMMARY

The computer is a tool for solving problems. A modern computer has multiple layers:

Hardware

BIOS

Compiler

Operating system

Application programs

The hardware has five types of functional units: memory, arithmetic and logic unit (ALU), control unit, input unit and output unit. The ALU and control unit are together known as Central Processing Unit (CPU) or processor. The memory and CPU consist of electronic circuits and form the nucleus of the computer. The input and output units are electromechanical units. They are also known as peripheral devices.

A program is a sequence of instructions for solving a problem. All modern computers use the stored program concept. The program and data are stored in a common memory. The control unit fetches and executes the instructions in sequence one-by-one. A program can modify itself; instruction execution sequence also can be modified.

The main memory is the one from which the CPU fetches the instructions. The total capacity of a memory is the number of locations multiplied by the word length. The time taken by the memory to read the contents of a location (or write information into a location) is called its access time. The main memory has to be a (Random Access Memory) RAM that has equal access time for all locations. For a CPU with n address bits, 2^n is its memory addressability.

In a modern computer, usually a small part of the main memory is a Read Only Memory (ROM). Some permanently required control programs and BIOS are stored in ROM by the manufacturer.

The auxiliary memory is external to system nucleus and it can store large amount of programs and data.

A peripheral device is linked to the system nucleus by a device controller (I/O controller). Its main function is transfer of information between the system nucleus and the device. Functionally, a device controller communicates with a device through the device interface that carries the signals between the device controller and the device. All device controllers communicate with CPU or memory through the system interface. An I/O driver for a device consists of routines for performing various operations: read, write etc.

Every instruction specifies a macrooperation to be performed by opcode field. To execute an instruction, the CPU has to perform several microoperations. The microoperations are elementary operations. The clock signal is used as a timing reference by the control unit. The clock signal is a periodic waveform since the waveform repeats itself. The CPU fetches one instruction at a time, executes it and then takes up next instruction. This action is known as instruction cycle. In running state, the CPU performs instruction cycles and in halt state, it does not.

An interrupt is an event requiring some urgent action. In response to the interrupt, the CPU branches to an interrupt service routine.

An I/O routine can follow three different methods for data transfer: program mode, interrupt mode and DMA mode. In program mode (polling), the I/O routine (hence the CPU) is fully dedicated to data transfer from beginning to end. In interrupt mode, the CPU switches between an I/O routine and some other program in-between transfer of successive bytes. Data transfer from high speed devices such as hard disk and floppy disk is difficult with programmed mode or interrupt mode since transfer rate of these modes is slower than the rate at which data is coming from these devices. The DMA mode is used for high speed devices. The DMA controller controls the transfer of data between memory and I/O controller without the knowledge of the CPU.

Any computer that is designed around a microprocessor is a microcomputer. In micro computers, bus concept is used for inter connection of signals between the subsystems. The bus is a shared common path for several sources and destinations. Microprocessors have two different bus structures: internal bus and external bus. A sequence of events performed on the bus for transfer of one byte through the data bus is called a bus cycle. An I/O port is a program addressable unit which can either supply data to the CPU or which can accept data given by the CPU. Functionally, a port serves as a window for communication with CPU by interfacing the port hardware through data bus.

The process of loading the operating system into main memory immediately after powering on a computer is called booting. The boot strap loader is a small program in ROM which helps in booting.

The performance of a computer is measured by the amount of time taken to execute a program. Benchmark programs are evaluation programs used to compare the performance of different computers. In the early days, the term MIPS was used to indicate the speed of a computer. The SPEC is a non-profit organization dedicated for performance evaluation. It selects/develops typical application programs for different application areas and publishes the performance results for important commercial computers. The performance of a commercially available computer is taken as the reference computer. The performance of other computers are rated as a relative measure of the standard computer. Several techniques have been developed over the years to increase the system performance.

A computer's structure indicates its internal interconnections. The functional structure identifies functional blocks and the relationships between these blocks. The physical structure identifies the physical modules and the interconnections between them.

A computer's function indicates its behavior. The output of a computer is function of its input. At the overall level (i.e., as a system), its function is program execution.

A computer's architecture is defined as the attributes of the computer system that are needed by a machine language programmer/system software designer to develop a program. Computer organization gives an in-depth picture of the functional structure of a computer and logical interconnections between different units (functional blocks).

While designing a computer, first its architecture is fixed and then its organization is decided. The architecture defines the overall features of a computer whereas, the organization decides its performance. The implementation of a computer's architecture (and organization) is in the form of hardware circuits grouped into functional blocks. Physical implementation of a computer design depends on the component technology.

REVIEW QUESTIONS

1. A desktop PC is bigger in size and consumes more power than a notebook PC. Why then is the notebook costlier than the desktop? (Hint: portability).
2. The performance of two different computers A and B (having similar architecture) are being compared by a consultant as part of evaluation process. Computer A operates at 100 MHz clock and gives 100 MIPS whereas computer B operates at 120 MHz clock and gives 80 MIPS. Due to various reasons, computer B was chosen by the consultant. He also came out with few suggestions for improving the performance of

computer B in future design modifications. Some of his suggestions are given below:

- (a) Replace the existing main memory with a faster memory.
- (b) Introduce a small cache memory.
- (c) Increase the clock frequency to 200 MHz.

Suppose you are asked to select only one of these suggestions keeping the cost as the main factor which one will you select (a, b, c)? Which one needs a change in architecture? Which one needs better technology?

3. Semiconductor memory is small in size and faster than core memory. Also its power consumption is less than that of core memory. Both are random access memory (RAM). Still core memory is superior to semiconductor memory in only one aspect. What is that?

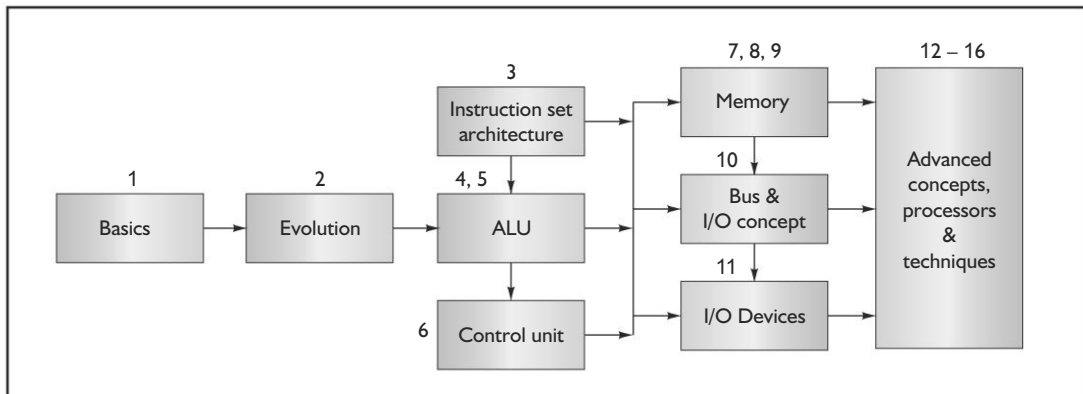
EXERCISES

1. The Intel Pentium processor has 32-bit address and its memory is byte addressable. What is the maximum physical memory capacity for a microcomputer built around this microprocessor?
2. The Intel 80286 microprocessor has 24-bit address. What is the maximum number of 32-bit words possible for its memory?
3. The original IBM PC based on Intel 8088 supplied a clock signal of 4.77 MHz to the microprocessor. What is the period of the clock signal?
4. A benchmarking program of 200,000,000 instructions was run on a computer operating at 100 MHz. It was found that the computer offered a performance of 100 MIPS. Determine the CPI, the cycles per instruction, for this computer?
5. A benchmarking program with following instruction mix was run on a computer operating with a clock signal of 100 ns: 30% arithmetic instructions, 50% load/store instructions and 20% branch instructions. The instruction cycle time for these types are 1, 0.6 and 0.8 μ s, respectively. Calculate the CPI (cycles per instruction) for this computer.

CHAPTER 2



EVOLUTION OF COMPUTERS



Chapter Flow Diagram

OBJECTIVES

- ◄◄ *Evolution of computers*
- ◄◄ *History of computers*
- ◄◄ *Computer generations*

KEYWORDS

CISC

RISC

SSI (Small Scale Integration)

MSI (Medium Scale Integration)

LSI (Large Scale Integration)

VLSI (Very Large Scale Integration)

Microprogramming

Multiprogramming

Timesharing

Batch Processing

Cache Memory

Virtual Memory

Abacus

Stored Program Concept

IAS (Institute for Advanced Studies) Computer

Workstation

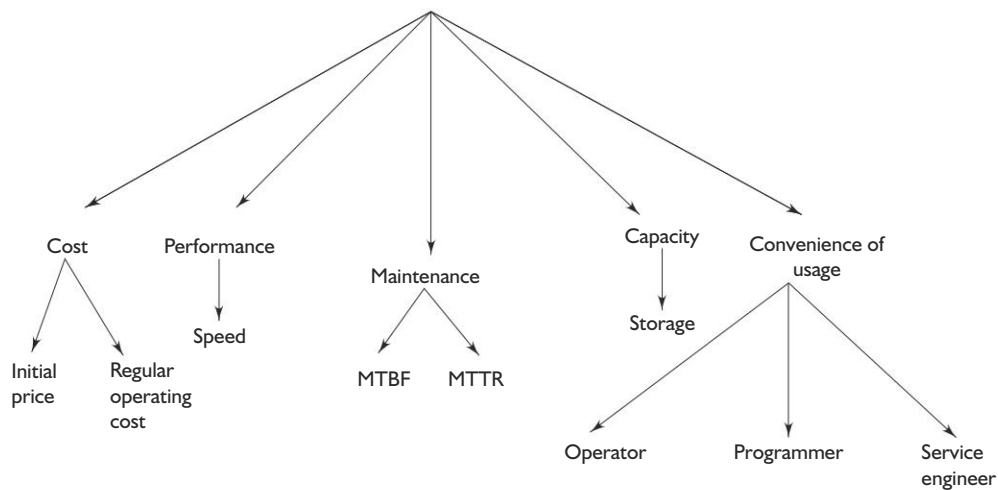
2.1 Introduction

The process of designing a computer requires sophisticated and scientific methodology. Over the years, several techniques have been developed towards this goal. Some of these have become obsolete due to new innovations whereas others still continue to be used.

This chapter reviews how the computer has evolved over the past sixty years. The details covered in this chapter will help the readers appreciate the concepts and techniques of the computer design covered in following chapters.

2.2 Dimensions of Computer Evolution

There are five dimension to measure the computer's excellence. These are Performance, Capacity, Cost, User friendliness and Maintainability. Figure 2.1 illustrates the important issues involved in these dimensions. These issues are considered in designing the computer.



MTBF—Mean time between failures; MTTR—Mean time to repair

Fig. 2.1 Five dimensions of importance

2.2.1 Contributing Factors

As shown in Fig. 2.2, three factors decide the superiority of the computer: Technology, Concepts and Techniques. The technology is constantly evolving in every aspect of the computer such as: CPU, memory, peripheral device, interconnecting path, software, etc. The concepts followed in the design of a computer decide its performance and capacity. Computer designer uses latest technology for designing the computer. Usually more than one design technique are available for implementing a concept.

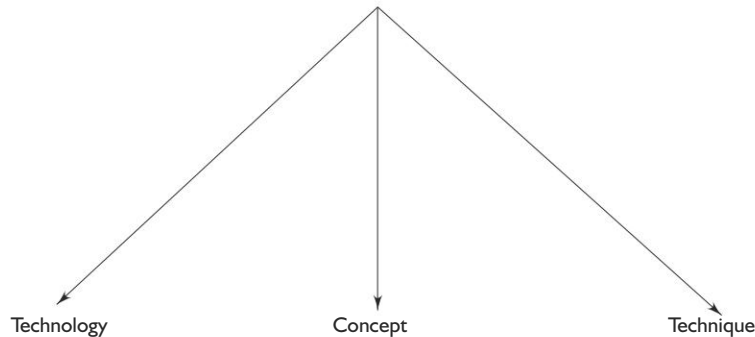


Fig. 2.2 Factors of design superiority

2.2.1.1 Usage Mode of Computer

Figure 2.3 shows different modes of usage of the computer system popular at different times. In each mode, the operating system and its associated system software has some unique features. For example, in a multiprogramming system, the operating system ensures protection of each program without interference from others.

2.2.1.2 Basic CPU Architecture

Figure 2.4 shows three different modes of CPU architecture. In *register architecture*, the operands for the instructions are put into the CPU registers and hence they are fetched fast during instruction cycle. In *accumulator architecture* the number of instructions in the program increases but the execution of each instruction is fast since one operand is in the accumulator itself. In *stack architecture*, programming is very simple since any arithmetic operation is done on the item at the top of the stack. The CISC architecture supports both simple and powerful instructions whereas the RISC architecture supports only simple instructions.

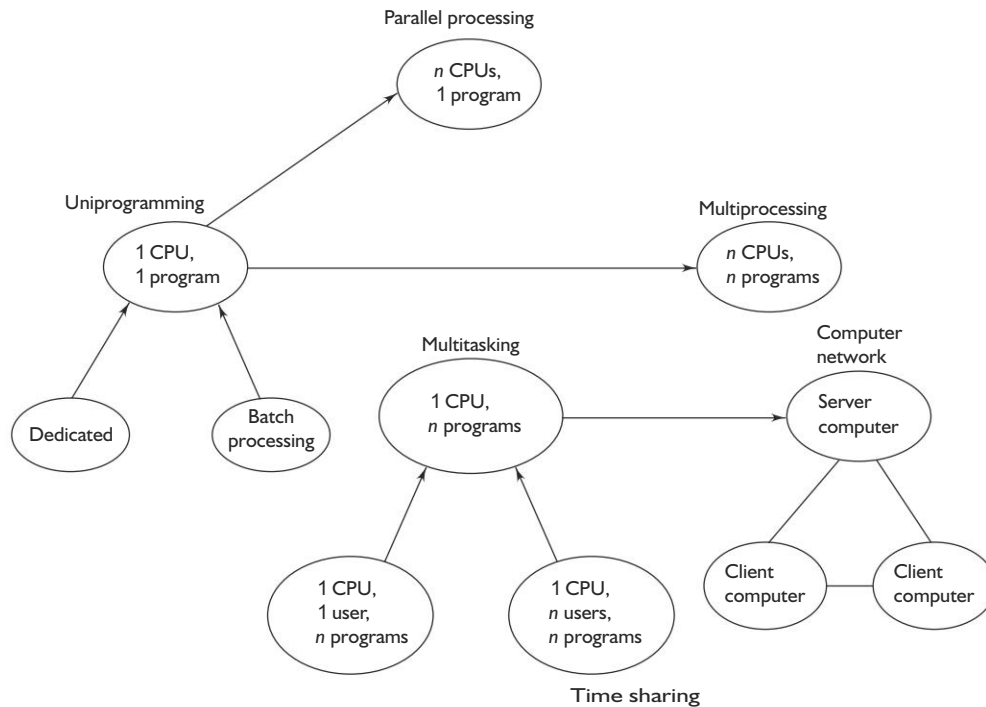


Fig. 2.3 Evolution of mode of usage

2.2.1.3 Storage Styles

Figure 2.5 shows different types of storage in a computer. The registers within CPU also serve as storage but they store only the operands and not the instructions. Its capacity is also limited. Hence, the registers are not included while talking about the memory of a computer. The capacity of the main memory is important since it decides the size of the active program at any given time. Since the main memory is costly, the designers provide cheaper secondary storage devices which function as an extension of the main memory for storing programs and data. This forms a two level storage. A large capacity of secondary memory (disk, tape etc.) is provided. The access time of secondary memory is more than that of main memory. In a three level storage, a small but high speed cache memory is used as a temporary buffer between the CPU and main memory. It is used to store and supply the contents of frequently accessed main memory locations.

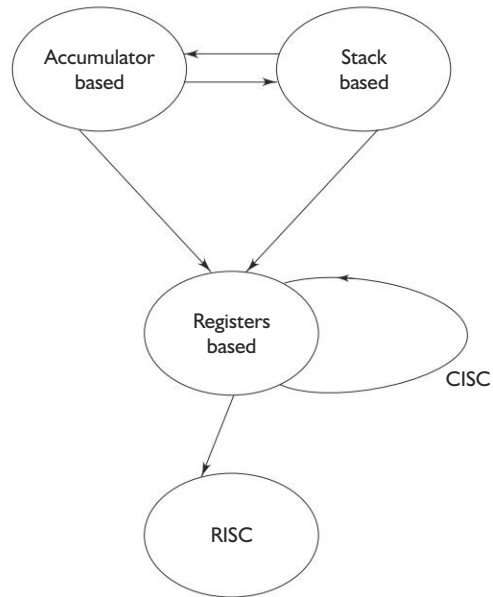


Fig. 2.4 Modes of CPU architecture

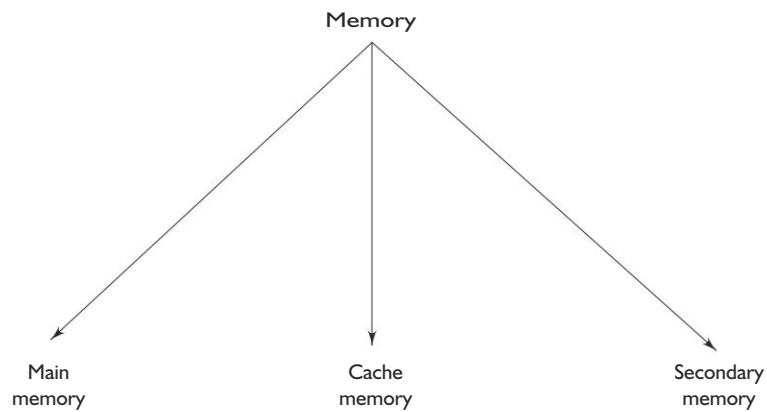


Fig. 2.5 Storage types in computers

The software recognises the registers and uses them for storing the operands of instructions. Whereas, the cache memory is an internal feature of CPU and the software does not consider it as a memory unit at all. In fact, the cache memory is not an additional storage; it is the buffer which increases the performance.

2.2.1.4 Control Unit and Instruction Decoding

Figure 2.6 shows the trends in instruction decoding in the control unit. The hardwired control unit has hardware circuits which decode the opcode in the instruction and generate appropriate control signals. The microprogrammed control unit stores the required control

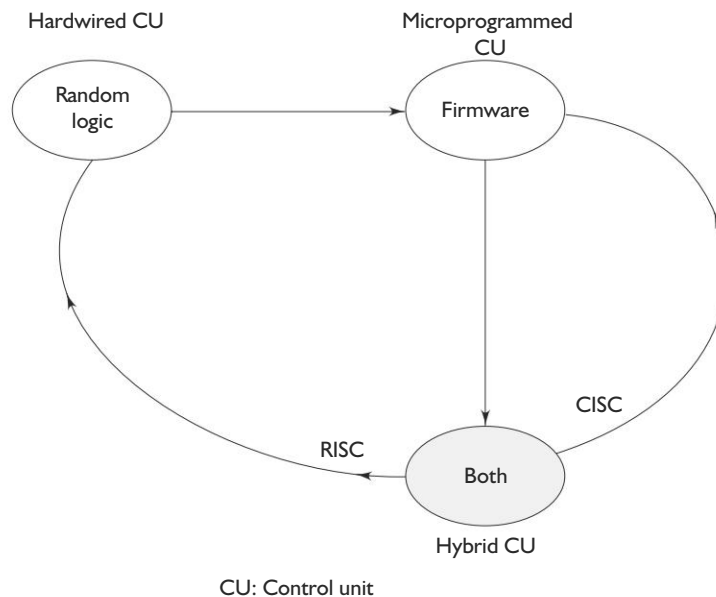


Fig. 2.6 Evolution of control unit design—instruction decoding

signals as microinstructions for each instruction. A read only memory inside the control unit stores the *microprograms* (sequence of micro-instructions) for each instruction. Thus, the control unit serves as a 'stored microprogram' sequencer.

The development of microprograms is easier while designing the control unit. But the CPU operation is slow since the control signal patterns have to be fetched from the read only memory. The hybrid control unit uses a mixture of both ideas. Part of the control unit which

is time critical is hardwired and the remaining part is microprogrammed. The RISC system has to be as fast as possible and hence usually its control unit is hardwired.

2.2.1.5 Main Memory Techniques

Figure 2.7 shows some of the memory techniques. CPU speed is constantly improving due to advanced technology but the memory technology has not evolved that rapidly. Hence, the aim is to get better performance from the same given technology for the memory. *Interleaving* is a concept of dividing the memory into two parts: odd addressed and even addressed. In other words, adjacent locations are placed in separate modules which can be accessed simultaneously. This reduces the overall access time.

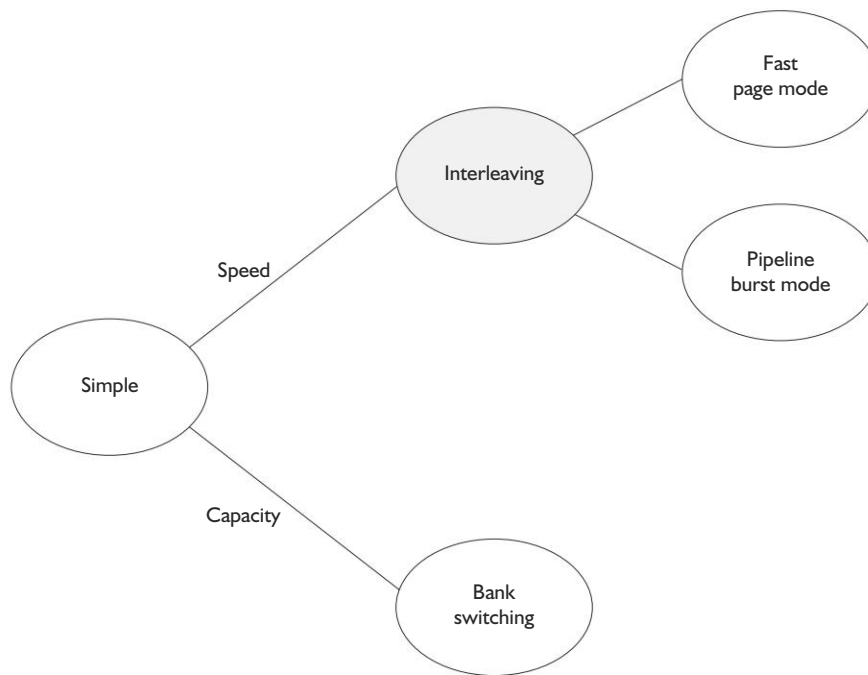


Fig. 2.7 Evolution of main memory concepts

For a given CPU, the memory capacity is limited by the number of address bits. The bank switching is a concept of breaking this limitation without the knowledge of CPU. Multiple banks of same capacity is used and at a time one of them is selected. The operating system keeps track of the banks.

2.2.1.6 Instruction Cycle Handling

In general, the performance of a subsystem can be increased by multiple techniques as shown in Fig. 2.8. These are discussed in following chapters. Figure 2.9 shows different techniques of instruction cycle handling in order to increase the number of instructions processed per second.

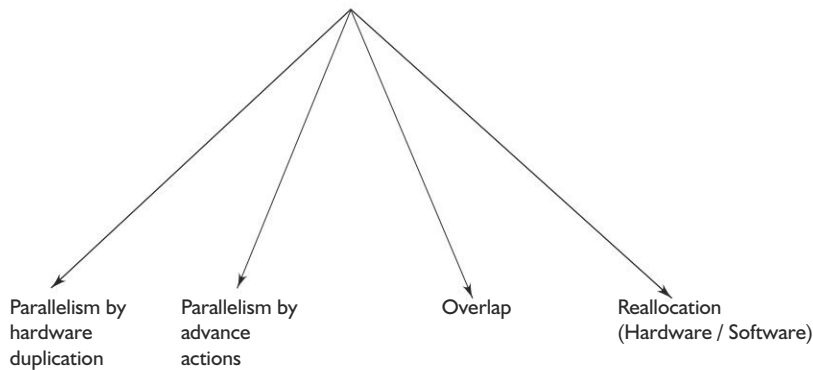


Fig. 2.8 Strategies for increasing performance

2.2.1.7 I/O Techniques

Figure 2.10 shows the different techniques of transferring data with peripheral devices. The choice made for each device is a trade-off between speed (transfer rate desired) and cost (hardware).

2.2.1.8 System Software

The operating system and other system software developments have grown parallel with the hardware and architecture. Figure 2.11 shows some important developments in system software. With each development, the programmer's role gets redefined. He concentrates more on algorithms than the machine related issues. However, some amount of overhead is added to program execution time with each new development. This time is negligible due to the increase in CPU speed and other techniques of parallelism and overlap. In embedded system, the entire system software gets embedded inside the hardware permanently.

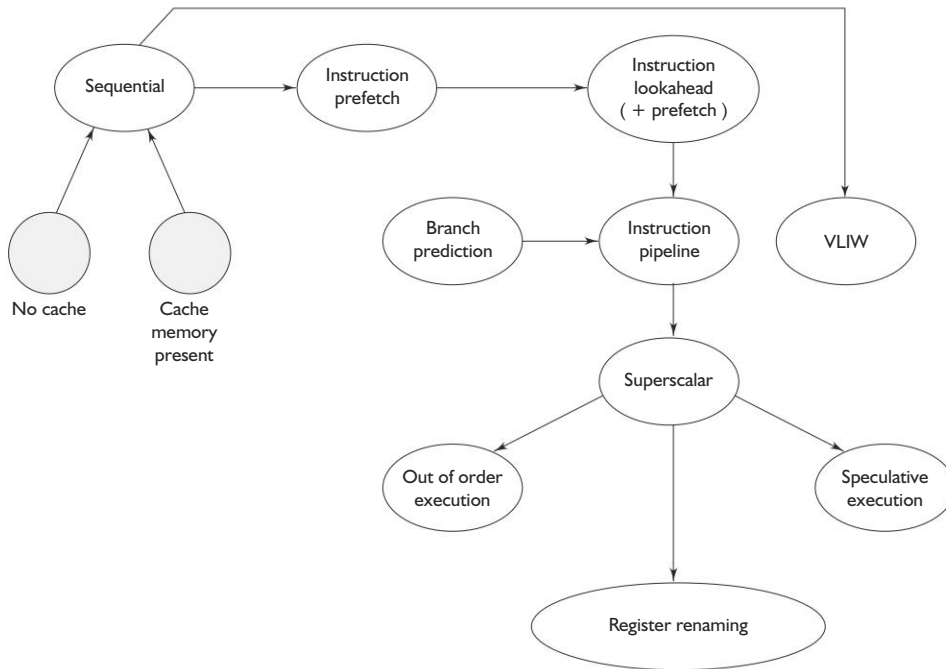


Fig. 2.9 Evolution of instruction cycle handling

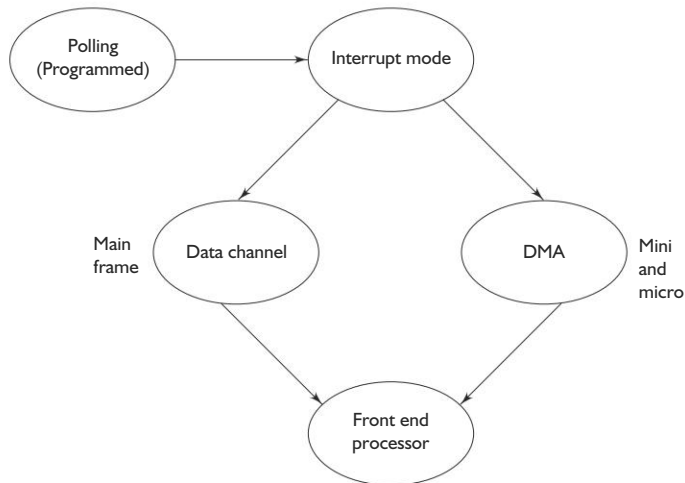


Fig. 2.10 Evolution of I/O techniques

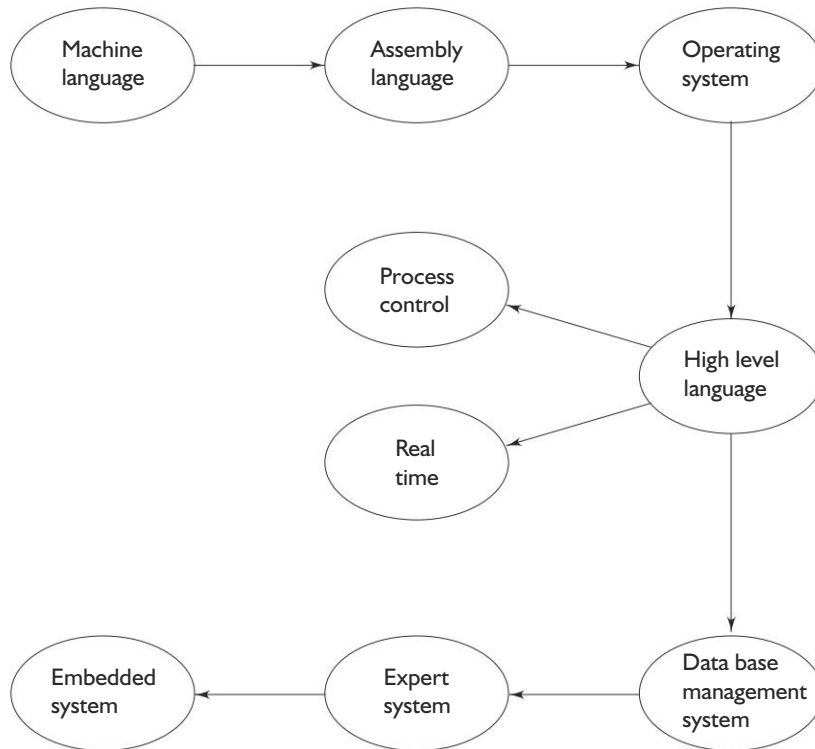


Fig. 2.11 Evolution of system software

2.3 History of Computers

The present day computer is the result of combined efforts of several scientists over the past 60–70 years. Early developments in computers were achieved at Universities working on Government funded projects. The requirement of high speed calculations by United States military contributed to many significant inventions in the early days whereas most of the developments in recent years are by private industry. The history of computers is covered by three distinct types of computers: Mechanical computers, Electro-mechanical computers and Electronic computers. The electronic computers are further grouped into different *computer generations*.

2.3.1 Early Calculators/Computers

2.3.1.1 Abacus

The *abacus* is a mechanical device used for counting. The mathematical operations such as addition, subtraction, division and multiplication can be performed on a standard abacus. The abacus is nearly 2000 years old. It is still being used by Chinese merchants. It is useful for teaching simple mathematics to children. The abacus (Fig. 2.12) has a wooden frame with vertical rods on which wooden beads slide. Arithmetic problems are solved when these beads move around (with the fingers) according to programming rules. Beads are considered counted, when moved towards the beam that separates the two decks: upper deck and lower deck. Each bead in the upper deck has a value of five; and each bead in the lower deck has a value of one.

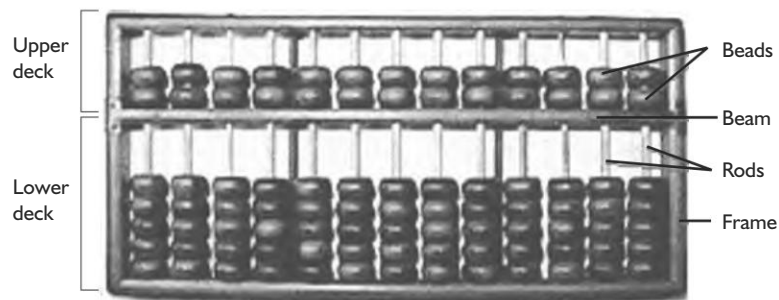


Fig. 2.12 Abacus

2.3.1.2 Mechanical Computers/Calculators

Mechanical calculators were based on simple parts such as gears and levers. The French philosopher Blaise Pascal built the first digital computer in 1642. It performed addition and subtraction of decimal numbers. In Germany, Gottfried Wilhelm von Leibniz invented a computer that could add, subtract, multiply and divide. Leibniz designed a stepped gear mechanism for introducing the addend digits.

A sophisticated calculator was designed by Charles Xavier Thomas. It could add, subtract, multiply, and divide. This was followed by models with a range of improved capabilities such as accumulation of partial results, storage and automatic re-entry of previous results and printing of the results.

Babbage

In 1822, Babbage, a mathematics professor, designed a model for an automatic mechanical calculating machine, called a difference engine. It was planned to be steam powered and fully automatic, including the printing of the tables. The British government provided financial help for developing it. Before completing the difference engine, Babbage got a better idea. He proposed a general purpose, fully program-controlled, automatic mechanical digital computer, called an Analytical Engine. It was a decimal computer with word length of 50 decimal digits and a storage capacity of 1,000 digits. Conditional branch instruction was an interesting feature on this machine. But this his new proposal was not accepted by the government.

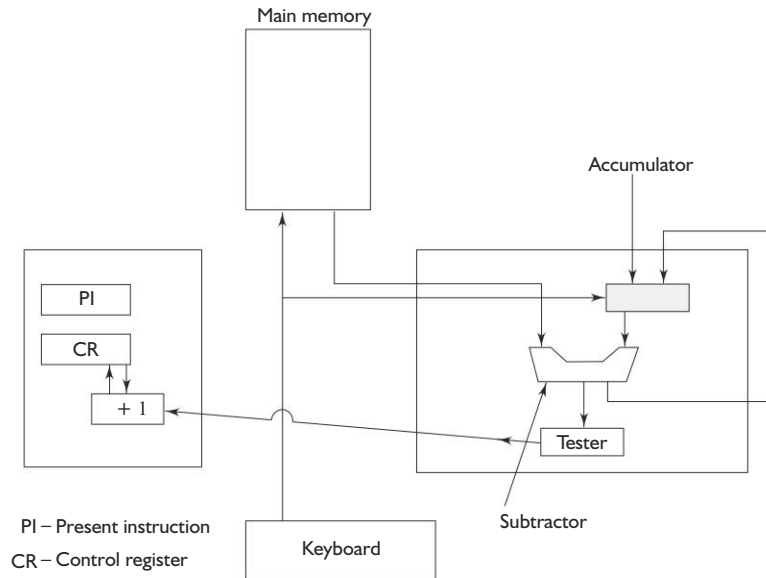
Between 1850 and 1900, many inventions were made in mathematical physics. It was discovered that most observable dynamic phenomena can be defined by differential equations. The construction of railway tracks, steamships, bridges etc. required differential calculus. These accelerated, the research for inventing a machine that could rapidly perform repetitive calculations.

2.3.1.3 Punched Cards

The development of punched cards by Herman Hollerith and James Powers was a significant contribution to development of computers. They introduced offline equipments to read punched information (holes) on the cards. This facilitated faster input of the data and the program. Also, the decks of cards were used as offline secondary storage. International Business Machines (IBM), Remington, Burroughs, and other companies introduced punch card systems with electromechanical devices.

2.3.2 Manchester Mark I

The Mark I stands between electro-mechanical computer and electronic computer. Some people consider it to be the world's first stored program computer since it was successfully demonstrated in June, 1948. Williams and Kilburn were the main architects of this machine. They developed a Cathode Ray Tube (CRT) based storage which allowed random access. A serial subtractor was the heart of the arithmetic unit. It was built using an expensive Pentode. All other registers used Williams CRT tubes. Figure 2.13 gives an overview of Mark I computer. The instruction has provision for one operand field only. The accumulator contents works as the second operand. The following six instructions are supported by Mark I:


Fig. 2.13 Mark 1 organization

1. Subtract
2. Load negative
3. Store
4. Test for zero
5. Jump (indirect)
6. Jump (relative indirect)

Table 2.1 lists the features of Mark 1.

TABLE 2.1 Mark 1 features

S. no.	Name of the features	Mark 1
1	Arithmetic	Serial binary; 2's complement
2	No. of instructions	6
3	Instruction format	Single address
4	Instruction length	16 bits
5	Instruction execution time	1.2 ms
6	Word length	32 bits
7	Main memory type	CRT based RAM
8	Memory capacity	32 words expandable to 8192 words
9	Supported peripherals	Keyboard, CRT monitor

2.3.3 Computer Generations and Evolution

The evolution of the computer over the period of time has resulted in development of various generations. Different technologies have been used for manufacturing the computer hardware. Based on the component technology, the computer is classified into five generations, as listed in Table 2.2. Though, the main basis for the classification is the component technology, each new generation of computers has a different architectural structure as compared to the earlier generation. The study of these aspects, helps one to distinguish between the past and the present dimensions of the computer.

TABLE 2.2 Computer generations

Generation No.	Technology	Duration	Popular computers	Major new inventions
1	Vacuum tubes	1945–1958	Mark1, ENIAC, EDVAC, UNIVAC I, IBM 650, IBM 701	Stored program concept, magnetic core memory as main memory, fixed point binary arithmetic
2	Transistors	1958–1966	ATLAS, B 5000, IBM 1401, ICL 1901, PDP-1, MINSK-2	Operating system, multi-programming, compiler, magnetic hard disk, floating point binary arithmetic, minicomputer
3	Integrated circuits (SSI and MSI)	1966–1972	IBM system/360, UNIVAC 1100, HP 2100 A, PDP-8	Multiprocessing, semiconductor memory, virtual memory, cache memory, supercomputer
4	LSI	1972–1978	ICL 2900, HP 9845 A, Intel 8080	RISC concept, microcomputer, process control, workstation
5	VLSI	1978 onwards	IBM RS/6000, SUN Micro Systems; Ultra SPARC family	Networking, server system, multimedia, embedded system

2.3.3.1 First Generation Computers

The following computers are significant milestones of first generation computers:

1. Electronic Numeric Indicator and Computer (ENIAC)
2. IAS

3. UNIVAC I
4. IBM 701

The main contributions of first generation computers are listed below:

1. Use of vacuum tubes for processing and storage
2. Common high speed memory for program and data
3. Use of fast main memory and slow secondary memory
4. Use of input–output instructions
5. Introduction of ferrite core memory
6. Introduction of assembly language to avoid tedious machine language programming
7. Use of electromechanical magnetic drum as secondary memory
8. Use of registers for storing operands and results of instructions inside CPU
9. Use of peripheral devices such as magnetic tape, magnetic drum, paper tape and card punch
10. Use of interrupt concept

Operation of First Generation Computers

The first generation computers were pure hardware machines. They did not have operating system. Programming was done in the machine language which differs from one type of computer to another. The user deals with several switches in the front panel to start, run or halt the computer. The internal status of the computer is displayed on several lights on the front panel. Invariably only a designer or programmer could operate the computer due to the complexities involved.

(i) ENIAC

Table 2.3 identifies the significant aspects of the ENIAC. It was developed at the University of Pennsylvania to handle the ballistic tables for the U.S. Army. It was a decimal computer with a set of accumulators. It was thousand times faster than the relay computers. Programming the ENIAC was tedious task as it involved manual setting-up of switches and cables. To input a program, several hours of manual work was needed. The ENIAC was the first popular electronic computer. It was used in World-War II for automatic calculation of the ballistic tables, but it was publically declared only in 1946. The difficulties faced in programming ENIAC led to the birth of stored program concept. In 1945, John Von Neumann, who was a consultant to the ENIAC presented a new project called the Electronic Discrete variable Computer (EDVAC). ENIAC multiplied two decimal numbers by reading the product from the multiplication table in memory. ENIAC is commonly accepted as the first successful high speed Electronic Digital Computer (EDC) and was used from 1946 to 1955.

TABLE 2.3 ENIAC features

S. No.	Name of the feature	ENIAC
1	No. of vacuum tubes	18000
2	Power consumption	140 kW
3	Floor space	1800 sq. ft
4	Arithmetic	Decimal
5	Word length	10 digits
6	Main memory type	Separate memories for program and data
7	Memory capacity	20 × 10 digits
8	Speed	5000 additions/sec
9	Major operations	Addition, subtraction, multiplication, division, square root calculation
10	Supported peripheral devices	Punch card, electric typewriter

(ii) EDVAC AND STORED PROGRAM CONCEPT

The basic principle was that a computer should have a very simple, fixed physical structure, and be able to execute any kind of computation by means of a proper programmed control without modifying the unit. The EDVAC has the first stored program concept, that has three main principles:

1. Program and data can be stored in memory.
2. The computer executes the program in sequence as directed by the instructions in the program.
3. A program can modify itself when the computer executes the program.

The EDVAC was a binary computer with 1-bit adder. It had a two tier memory hierarchy:

1. A fast main memory of 1 K words.
2. A slow secondary memory of 20 K words.

The instruction format used three addresses:

1. Two addresses for operand storage.
2. One address for result storage.
3. One address for indicating next instruction address.

ENIAC supported the conditional branching instruction in which two way branching is specified by the third and fourth addresses. It also had input–output instruction to perform data transfer. The bit-by-bit processing was slow but it reduced the hardware cost.

(iii) IAS Computer (Von Neumann Machine)

The IAS computer was developed at the Princeton Institute for advanced studies. It is the

basic model for the stored-program concept which is followed in almost all subsequent computers. The project team was led by *Von Neumann*.

In the IAS computer, the instruction has two fields, opcode and address, as shown in Fig. 2.14. A memory word stores two such instructions. During instruction fetch, two instructions are fetched from the memory in one access. Figure 2.15 gives the organization of the IAS



Fig. 2.14 IAS instruction

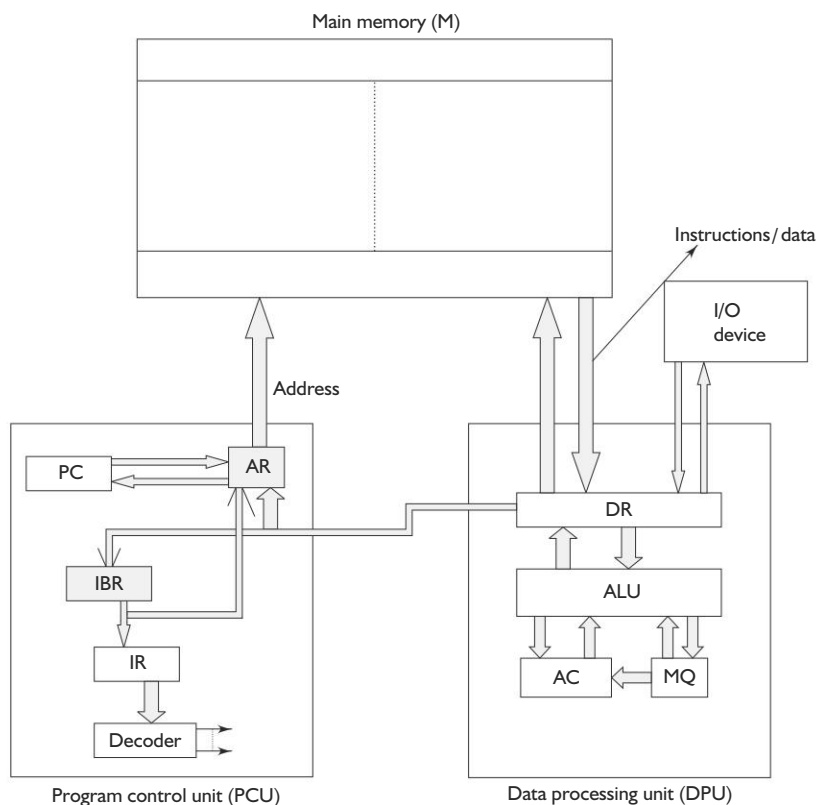


Fig. 2.15 IAS organization

computer and Table 2.4 lists the features of IAS. The instruction set has five types of instructions: Arithmetic, Data transfer, Unconditional branch, Conditional branch and Instruction address modify. The registers AC and MQ are directly addressable by instructions. The accumulator (AC) is a general purpose register whereas the MQ is special purpose register. The AC is used in most of the arithmetic and data transfer instructions since the AC contents plays a source, a destination or both in these instructions. The multiplier quotient (MQ) holds one operand (multiplier) for a multiply instruction and the remainder for a divide instruction.

TABLE 2.4 IAS computer features

S. No.	Name of the Feature	IAS computer
1	Arithmetic	Binary; fixed point
2	No. of instructions	21
3	Instruction format	Single address
4	Instruction length	20 bits
5	Memory capacity	1 K words expandable to 4 K words
6	Memory word length	40 bits
7	Secondary memory type and capacity	Magnetic drum; 16 K words

The program counter (PC) is incremented by 1 after every instruction fetch. Since two consecutive instructions are fetched simultaneously, the IR and IBR are used to store the current and next instructions, respectively.

Plus points of IAS computer are as follows:

1. The IAS computer is a single address machine. The short instruction length results in reduced program size and hence, main memory requirements. This leads to reduction in system price.
2. During instruction fetch, simultaneously, two instructions are brought from the memory. Thus, one instruction is always pre-fetched. This reduces access time for the second instruction, thus speeding up the instruction cycle time.
3. The use of 'address modify' instruction results in changing the address field of another instruction in the memory. Thus, a program can modify itself during runtime. This dynamic change is a powerful feature, resulting in the efficient programming.

Minus Points of IAS computer are as follows:

1. The IAS computer was weak in performing I/O operations. The input or output instruction results in transferring of data between the input device and memory or between the memory and output device. In both cases, the data had to pass through DPU, since there is no 'direct memory access' between memory and I/O subsystems.
2. The IAS computer did not have 'CALL' and 'RETURN' type of instructions. Hence, subroutine facility was not possible.

(iv) Beyond IAS

Universal Automatic Computer (UNIVAC I) was developed by Eckert–Mauchly Corporation. It was suitable for both scientific and commercial applications. This was soon followed by UNIVAC II which offered higher performance and had higher memory capacity. Subsequently, UNIVAC 1100 series of computers with compatibility between the various models was released.

International Business Machines (IBM) manufactured the IBM 701, its first electronic stored program computer, in 1953. It was designed for scientific applications. Soon IBM 702, a system designed for commercial applications was introduced. UNIVAC series and IBM series were two successful family of computers for a very long time.

2.3.3.2 Second Generation Computers

Invention of Transistor by Bell Labs was a boon to second generation computers. It was smaller in size and also consumed less power. Several companies such as IBM, NCR, RCA etc. quickly introduced transistor technology which also improved reliability of computers. Instead of *wiring* circuits, photo printing was used to build Printed Circuit Boards (PCB). This modularized the computer hardware into a set of easily replaceable functional circuit boards. Both computer production and maintenance of computers became easier. The use of high level programming language is another major improvement in the field of second generation computers. Computer manufacturers have also developed compilers for different programming languages such as FORTRAN and COBOL. The second generation computers had a variety of peripheral devices such as console typewriter, card reader, line printer, CRT display, graphic device etc. Newer application programs were available for accounting, payroll, inventory control, purchase order generation, invoicing etc. Large organizations which installed computers also formed their own teams of programmers to develop in-house programs. Application of the computers in newer places such as libraries and hospitals became popular. Computerization spread to non-western countries also.

Table 2.5 lists several concepts and techniques associated with second generation computers.

TABLE 2.5 Features of second generation computers

S. No.	Name of the feature	Types	Remarks
1	Operating system	System software	Management of system resources and tackling user requirements are relieved from user (application) programs
2	Batch processing	System usage	Multiple programmers/users share the centralized large system by submitting their programs for the batch and collect the results later
3	Multiprogramming	System throughput improvement	Concurrent execution of multiple programs; multiplexing of CPU avoiding idle time during I/O operations
4	Timesharing	System usage	Multiple remote users share a single computer through their terminals; System allocates time slices to the terminal users offering fast response
5	High level programming language/compiler	Programmer aid	Simplification of computer programming; knowledge of hardware or machine language not needed for programming; programmer productivity improves
6	Magnetic hard disk	Auxiliary storage	Faster and more reliable than magnetic drum; flying read/write heads
7	Index register	Programmer aid	Used for operand addressing in iterations; efficient programming possible
8	CALL and RETURN instructions	Programmer aid	Subroutine facility is possible; repetitive programming work is avoided; programmer productivity as well as memory space utilisation improves
9	Floating point arithmetic	Dedicated ALU for floating point operations	Higher precision needed scientific applications
10	Data channel/DMA transfer	Dedicated hardware for data transfer	Supports high speed devices and also allows parallelism between CPU and I/O
11	Minicomputer	Low cost computer	Affordable computers for smaller organisations and institutions; reduced hardware than large system and reduced speed

2.3.3.3 Third Generation Computers

Invention of Integrated Circuit (IC) chip is a great event for electronics field giving rise to microelectronics. IC has multiple advantages over discrete components: smaller size, higher speed, lower hardware cost, improved reliability etc. Digital computer design became more attractive and interesting. The use of computers in a continuous processing and manufacturing sectors such as petroleum refining and electrical power distribution became popular. The computer families by leading companies such as IBM, UNIVAC, HP, ICL and DEC dominated the computer industry. Domination of minicomputers created more job opportunities to computer professionals. Table 2.6 lists the various concepts and techniques used in third generation computers. Birth of super computers created interest among research community.

TABLE 2.6 Features of third generation computers

S. No.	Name of the feature	Types	Remarks
1	Virtual memory	Cost reduction with limited physical memory	System manages running of larger programs with collaboration between CPU and operating system
2	Pipelining	Parallelism in instruction cycle	Overall throughput of CPU increases
3	Multiprocessing	Multiple CPUs in a single system	Simultaneous execution of multiple programs by different CPUs
4	Semiconductor memory	New memory technology with IC chips	Higher speed, smaller size and easy maintenance compared to core memory
5	Cache memory	Intermediate hardware buffer between CPU and main memory	Saving of CPU time (in instruction fetch/operand fetch) by supplying some instructions/operands from the buffer memory
6	Local storage	Internal registers in CPU	Operand fetch and result storing are faster; instruction cycle time reduces
7	Bus concept	New type of communication between CPU and other subsystems	Shared path; cost reduces; slower communication
8	Data communication	Communication between computers	Long distance data transfer through telephone lines
9	Micro-diagnostics	Maintenance aid; auto diagnostics	In-built diagnostic test routines stored in ROM perform auto testing of hardware

Leading computer manufacturers came out with families of compatible computers. IBM System/360 had a popular series with a range of models: 30, 40, 50 etc. The different models differed in capabilities and features (pricing also) but all the models were software compatible at the instruction set level. A newer model supported programs (applications and system programs) developed for older ones. UNIVAC, BURROUGHS and ICL were other companies who had their own series of compatible computers. There were also companies which manufactured 'IBM System/360 compatible' main frames which were cheaper than the equivalent IBM machines. The Amdahl and the EC series (1030, 1040, 1050, 1060, etc.) from Soviet Union and other communist countries are some of the classic examples.

2.3.3.4 Fourth Generation Computers

The LSI technology simplified the design of digital systems. Single-IC chip based computers came to market. The invention of microprocessor by INTEL gave birth to microcomputers. Several semiconductor companies such as Motorola, Fairchild, Texas Instruments and Zilog manufactured microprocessors that offer a fantastic range of capabilities. Several small companies also produce innovative computer design. Powerful *workstations* meant for specialised application such as CAD, production stations, test and repair jigs etc. were manufactured in addition to general purpose computers. Microcomputers slowly captured the market of mini-computers. Home computers and personal computers spread computer awareness and usage to new segments such as children and small businessmen. IBM PC and Apple PC gave new opportunities to both computer users and dealers. Since the IBM PC had 'open architecture', several small and medium manufacturers developed IBM PC 'clones' which were both hardware and software compatible to IBM PC. By hardware compatibility, it is implied that the hardware modules are changeable between different machines. Table 2.7 lists various concepts and techniques used in fourth generation computers.

TABLE 2.7 Features of fourth generation computers

S. No.	Name of the feature	Types	Remarks
1	RISC	Simple instruction set	Simplified control unit and increased parallelism achieve at least one instruction execution per clock
2	Workstation	Application specific computer	High speed systems for special applications; special hardware and software match the application type

(Contd.)

S. No.	Name of the feature	Types	Remarks
3	Microprocessor	Single-IC chip for CPU	Low cost micro computer challenging the minicomputer and spreading computer usage in all areas
4	Process control	Factory automation	Dedicated computers controlling the manufacturing process

2.3.3.5 Fifth Generation Computers

The use of VLSI and artificial intelligence concept is used in fifth generation computers. Expert systems, pattern recognition, voice recognition, signature capturing and recognition, microprocessor controlled robots etc. are some of the sophisticated developments in the field of computers. This leads to the rapid growth of computer professionals, trends and jargons in an uncontrolled way. Table 2.8 lists various new concepts and techniques used in fifth generation computers.

TABLE 2.8 Features of fifth generation computers

S. No.	Name of the feature	Types	Remarks
1	Portable computer	Senior executives carry even during travel	Special engineering offering light weight, battery operation and ruggedness facilitating usage even during travels
2	Networking	Computers linked to each other	Sharing of hardware/software resources and electronic communication
3	Server system	Fast and large capacity system	Saving of resources at client systems
4	Embedded system	Micro-controller based product	Dedicated intelligent controlling of equipment and tools including peripherals
5	Multimedia	Merger of data, sound, picture and voice	Newer applications of computers such as entertainment, education etc
6	Internet and email	Internet based usage	Everything from home is possible from learning to marketing

SUMMARY

A computer has to excel in five different dimensions: performance, capacity, cost, user friendliness and maintainability. Three factors contribute to the design superiority of the computer: Technology, Concepts and Techniques. The technology is constantly evolving in almost every aspect of the computer: CPU, memory, peripheral device, interconnecting path, software etc. The computer architect/designer makes use of latest technology at the time of designing a computer. The concepts followed in the design have a bearing on performance and capacity.

Early developments in computers were achieved at Universities working on government funded projects. The requirement of high speed calculations by United States military contributed to many significant inventions in the early days whereas most of the developments in recent years are by private industry. The history of computers is covered under three distinct types of computers: Mechanical computes, Electro-mechanical computers and Electronic computers. The electronic computers are further grouped into five different generations.

The stored program concept introduced by Von Neumann is the basis of almost any computer being designed today.

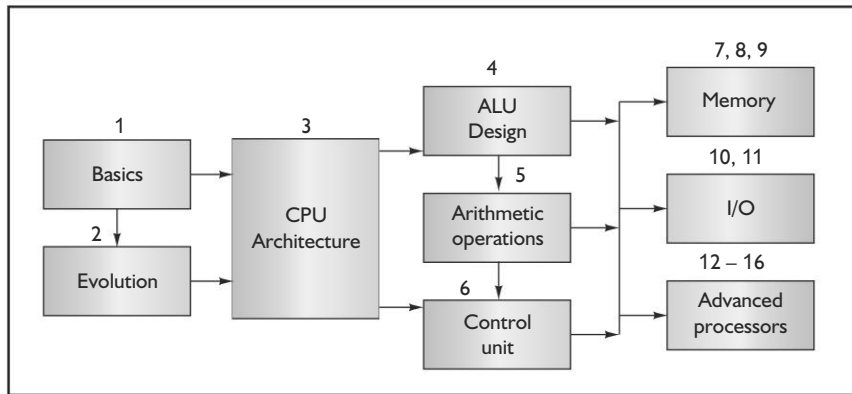
REVIEW QUESTIONS

1. A microcomputer is based on the INTEL 8080 microprocessor which has 64 KB memory space. The microcomputer has 128 KB physical memory. Guess the technique that would be used by the microcomputer designer to double the memory supported by the microprocessor. Note that the 8080 does not support virtual memory.
2. Intel 8086 and Intel 8088 are instruction set compatible microprocessors. The 8086 has 16-bit data bus whereas the 8088 has 8-bit data bus. The 8086 has 6-byte instruction (pre-fetch) queue whereas the 8088 has only 4-byte instruction queue. Do these two processors look alike to the software? In other words, do they have common architecture or different architectures?
3. Is IAS computer faster than Mark I computer? Is Mark I computer easy to program than IAS computer?

CHAPTER 3



CPU ARCHITECTURE AND INSTRUCTION SET PRINCIPLES



Chapter Flow Diagram

OBJECTIVES

- « *Instruction set architecture*
- « *Instruction types*
- « *Instruction sequencing*
- « *Hardware-software interface*
- « *Addressing modes*
- « *Operand types*
- « *CISC and RISC architecture*

KEYWORDS

Reduced Instruction Set Computing (RISC)	Immediate Addressing
Complex Instruction Set Computing (CISC)	Direct Addressing
LOAD/STORE Architecture	Register Addressing
Instruction Set	Condition Codes
Instruction Format	Big-endian
Addressing Modes	Little-endian
Relative Addressing	Macro
Index Addressing	Subroutine
Post-indexing	Fixed-point number
Pre-indexing	Floating-point number
Indirect Addressing	Stack

3.1 Introduction

This chapter discusses the overall architecture of the CPU. Several important parameters of the CPU have direct impact on performance of the system. The computer architect focuses on the design of the instruction set. Any weakness in the instruction set will drastically affect the machine language programmer and the compiler. This chapter discusses the various factors that influence the instruction set design. It also outlines the overall design approach for the processor.

3.2 Processor Role

The processor can be viewed at four different levels as shown in Fig. 3.1:

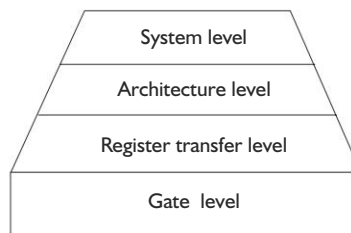
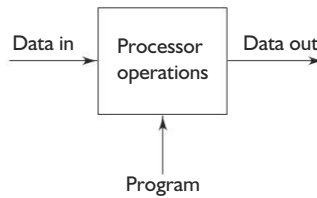
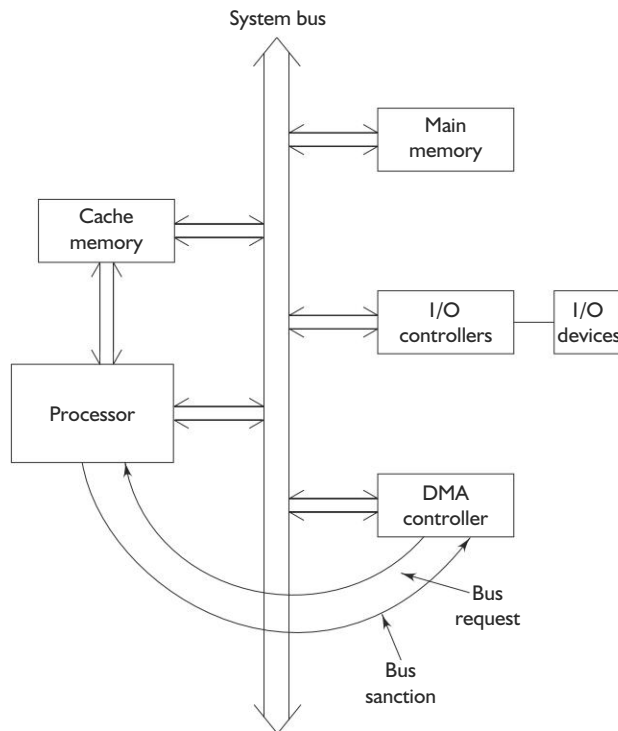


Fig. 3.1 Processor view

1. System level
2. Instruction set level or Architecture level
3. Register Transfer Level
4. Gate level

At the system level, the processor is the major subsystem of a computer. It has two functions:

1. *Program execution (performing instruction cycle)*: This involves a variety of data operations (Fig. 3.2) such as data processing, data storage, and data movement.
2. *Interfacing with other subsystems* such as Main memory, Cache memory, DMA controller, I/O controllers, etc. The processor has the responsibility of overall coordination of intercommunication between these subsystems. This is usually done through a bus structure as shown in Fig. 3.3.

**Fig. 3.2** Processor as a data operator**Fig. 3.3** Processor as a bus master

At the architecture level, the processor consists of program addressable hardware resources for the instruction set such as program counter, accumulator, operand registers, stack, interrupt vectors, status flags, etc. as shown in Fig. 3.4. Each instruction specifies a major role known as macro-operation. The exact macro-operation for an instruction is indicated in the opcode portion.

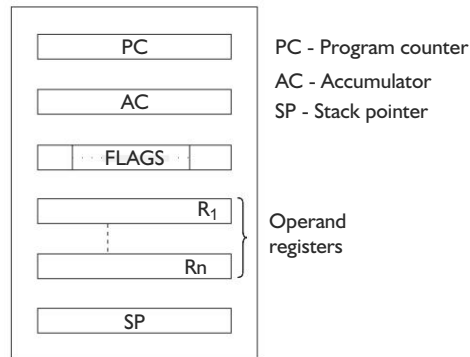


Fig. 3.4 Architecture level of processor

At the register transfer level, the processor has a set of digital modules such as registers, adders, counters, multiplexors, decoders, flip-flops, etc. each performing one or more specific functions. Each macro-operation involves one or more register transfer operations. For example, a MOVE macro-operation (i.e. moving the data from the memory location to the register) needs two register transfer operations (Fig. 3.5): memory read and register transfer.

At the gate level, the processor consists of hardware circuits which perform the micro-operations. In many practical cases, there is an overlap between register transfer level and gate level due to the usage of LSIs and VLSIs.

3.3 Processor Design Goals

The overall goal in designing a processor is to meet the needs of the instruction set. This involves designing appropriate hardware so that all macro-operations are executed accurately as per the requirements of the instructions. The designer has to consider certain other factors that are responsible for the success or failure of the processor in the commercial market. Figure 3.6 illustrates these factors that influence processor design. Certain important decisions made at the initial stage are as follows:

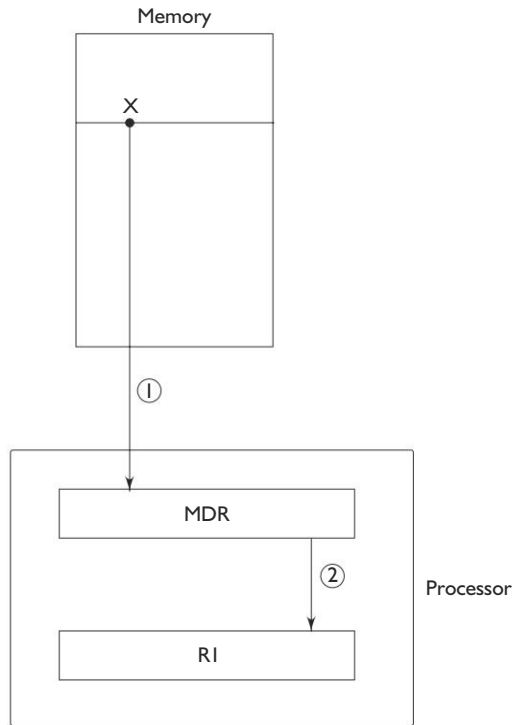


Fig. 3.5 Move macrooperation as two register transfers

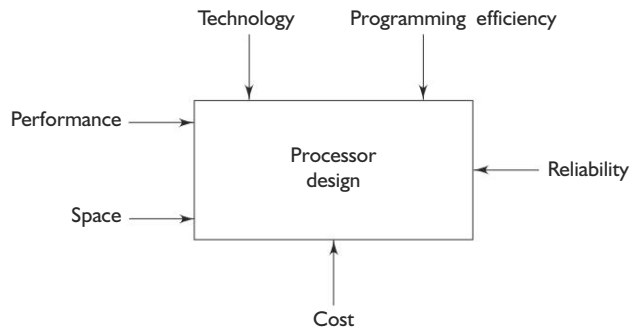


Fig. 3.6 Factors influencing processor design

1. Should floating-point instructions be executed by hardware or by software simulation?

2. What do we need:

- (a) A low cost processor with a single bus structure and limited hardware registers and other resources? or
- (b) A high performance processor with multiple bus and extensive resources and/or parallelism?

The extent of design varies with the type of processor. Figure 3.7 lists some processor types.

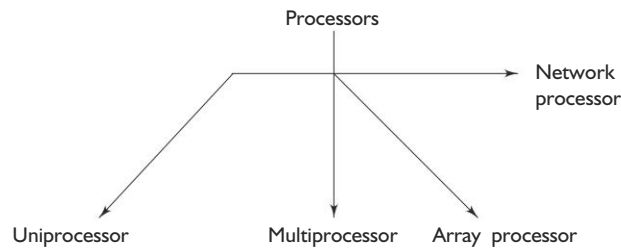


Fig. 3.7 Processor types

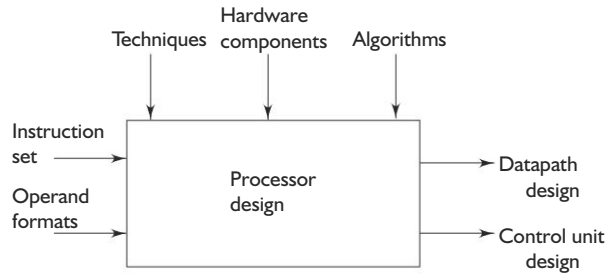
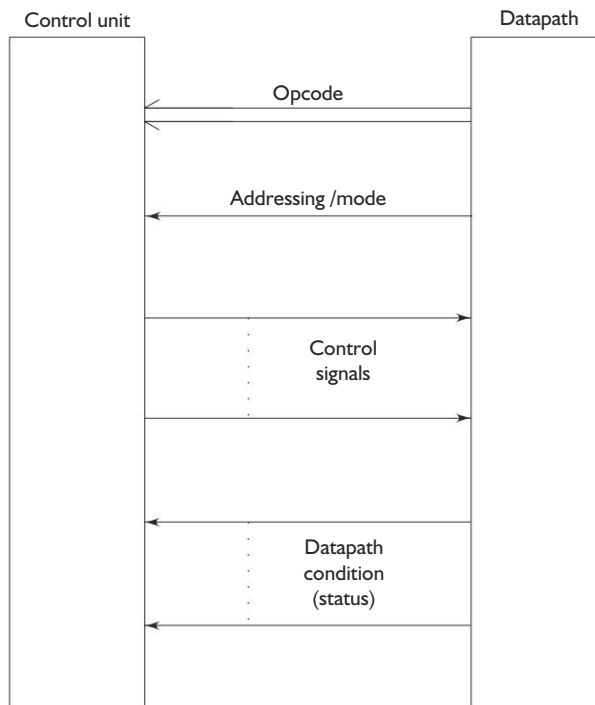


3.4 Processor Design Process

The design of a processor follows a series of steps each involving a different level of design activity. Some of the steps are interdependent and hence require some amount of *back and forth* travel. Figure 3.8 identifies the input and output of the processor design activity. There are two kinds of input. The first one (shown on the left side of the block) is finalized by the computer architect. It includes the instruction set description and operand formats. The second one (shown as the input from top) serve as a knowledge base. It includes the details of various algorithms, functions and behavior of different hardware components, and knowledge on different design techniques. The designer should evaluate the usefulness and limitations of the options available in each of these and pick the appropriate ones.

The dictionary of the hardware components keeps growing due to new inventions and developments. The designer should have a clear idea about the latest and obsolete components so as to come out with a state-of-the art design.

The processor is functionally divided into datapath and control unit. Figure 3.9 shows the interface between these two units. This interface is implemented in a low cost processor with a single bus. The same bus acts as an internal bus inside the datapath. In a medium

**Fig. 3.8** Input and output of processor design process**Fig. 3.9** Functional interface between datapath and control unit

performance based processor, two internal busses are included whereas, a high performance processor has three internal busses. Physically there is hardly any difference between the datapath and control unit. In a contemporary computer, the hardware of both the units are tightly coupled (integrated) in a single physical module as compared to the earlier main frame computers.

3.4.1 Processor Design Steps

The design of a processor is described in the following sequence of steps as indicated below:

1. Understand each instruction clearly. Define the macro-operation in terms of the given computer architecture.
2. Identify the hardware resources needed (to implement the macro-operation) in terms of programmer visible hardware items (registers, flags, stack etc.).
3. Translate each macro-operation as one or more register transfer operations.
4. Design the datapath (starting with the resources identified in step 2) required for performing the register transfer operations and identify the control points. Analyze whether the datapath circuits can be reduced by combining the designs for various instructions, eliminating redundant circuits.
5. List the timing sequence of control signals required to activate the control points.

3.4.2 Register Transfer Language (RTL)

The RTL is a notation used to specify the micro-operation transfer between the registers. It is a tool used for describing the behavior of the instructions. For example, the RTL statement $R3 := R1$ indicates a simple register transfer involving two registers R1 and R3. The action described by this statement is—the contents of the register R1 are transferred (copied) to the register R3. The same RTL statement can be also written as $R3 \leftarrow R1$. Figure 3.10 shows the datapath for this micro-operation.

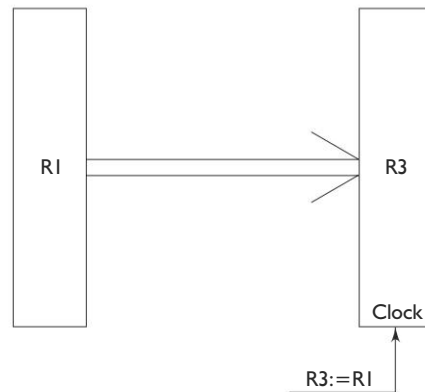


Fig. 3.10 Block diagram for register transfer

The micro-operations are classified into four types:

1. *Register transfer micro-operations*: transfer (copy) the contents of one register to another register without any change in contents.

2. *Arithmetic micro-operations*: perform the arithmetic operations on data present in the register: addition, subtraction, increment, decrement and shift are common arithmetic micro-operations. Table 3.1 describe them.

TABLE 3.1 Arithmetic micro-operations

S. No.	RTL notation	Description
1	$R5: = R1 + R3$	Contents of registers R1 and R3 are added and the sum is put in register R5
2	$R5: = R1 - R3$	Contents of R3 is subtracted from R1 contents and the result stored in R5
3	$R3: = \overline{R3}$	R3 content is complemented
4	$R3: = \overline{R3} + 1$	R3 content is converted into 2's complement
5	$R5: = R1 + \overline{R3} + 1$	R1 content is added with 2's complement of R3
6	$R1: = R1 + 1$	R1 content is incremented by 1
7	$R1: = R1 - 1$	R1 content is decremented by 1

3. *Logic micro-operations*: perform the bit manipulation operations on data available in registers. These micro-operations consider each bit as a separate binary variable. They are used for making logical decisions and also for bit manipulation of the binary data. The following special symbols are used for some of the logic micro-operations:

$\vee$: For logical OR

$\wedge$: For logical AND

$\oplus$: For exclusive OR

4. *Shift micro-operations*: perform the shift operations on data held in registers. These are of three types: logical shift operations, circular shift operations and arithmetic shift operations.

- *In logical shift*, all bits (including sign bit) take part in shifting operation. A '0' is entered in the vacant extreme bit (left most or right most) position as shown in Fig. 3.11.

- *In circular shift*, the bit shifted out from one extreme bit position enters the other extreme side as shown in Fig. 3.12. It is also known as 'rotate' operation.

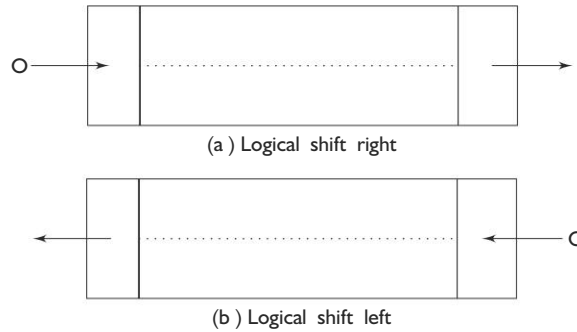


Fig. 3.11 Logical shift operation

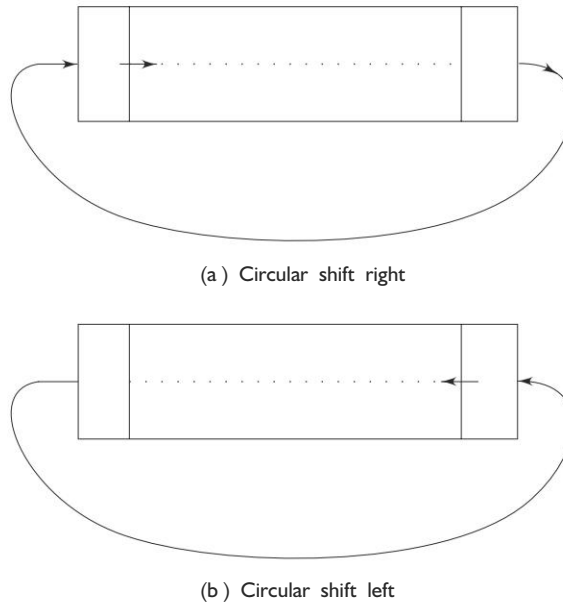
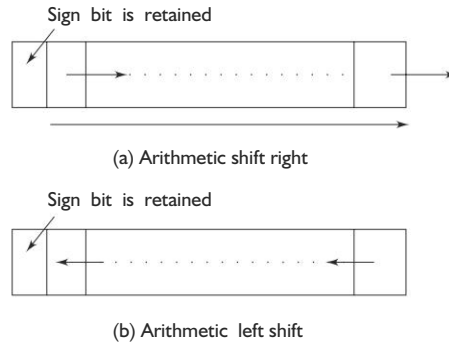


Fig. 3.12 Circular shift

- In *arithmetic shift*, sign bits remain unaffected and other bits take part in shift operation as shown in Fig. 3.13.

**Fig. 3.13** Arithmetic shift operation

3.5 Datapath Organization

The term datapath includes the following hardware components:

1. ALU.
2. Registers for temporary storage.
3. Various digital circuits for executing different micro-operations. These include hardware components such as gates, latches, flip-flops, multiplexers, decoders, counters, complementer, delay logic etc.
4. Internal path for movement of data between ALU and registers.
5. Driver circuits for transmitting signals to external units (control unit, memory, I/O).
6. Receiver circuits for incoming signals from external units.

3.6 Control Unit Organization

The control unit is the brain or *nerve center* of the computer hardware. It keeps track of the instruction cycle and generates relevant control signals at appropriate time so that appropriate microoperations are done in the CPU and other external units such as—memory and I/O controllers/devices. The control unit is designed for a specific datapath organization (ALU, registers etc).

The control unit is responsible for coordinating the activities inside the computer. Figure 3.14 provides a birds eye view of the overall role of the control unit. The control signals issued by the control unit reaches the various hardware logics inside the processor and other external units. These signals initiate different operations in the computer. The micro-operations are performed when the relevant control signals activate the control points. The main memory is controlled by two control signals: memory read and memory write. All I/O controllers receive the control signals IOR and IOW. The datapath receives maximum number of control signals. How does the control unit know when to issue which control signal? It behaves as dictated by the program which is being executed.

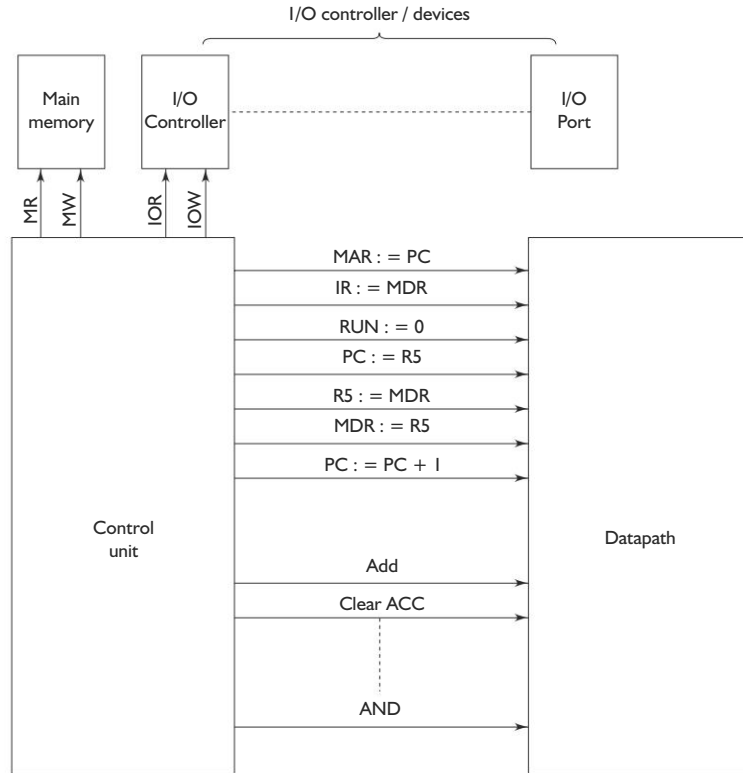


Fig. 3.14 Role of control unit



3.7 Instruction Sequencing

As seen in Chapter 1, there are multiple layers between a computer user's application program and the computer hardware. The user's program is executed under the control of the OS. To run an application program written in high level language, it is first translated into an object code by the compiler. Then the object code is stored in memory and executed by the CPU. The CPU executes any object program by performing instruction cycle repeatedly.

In Chapter 1, we have seen the actions done by the CPU for four basic types of instructions: ADD, JUMP, HALT and NOOP. The ADD instruction is a typical example for arithmetic instruction. The JUMP instruction is also known as branch instruction since it transfers program control (of CPU) from one place to another place in the program. When the CPU executes a program, the exact nature of actions done by the CPU for the program depends on the instructions in the program.

TABLE 3.2 Instruction types and CPU action

S. No.	Instruction type	CPU action	Remarks
1	Data transfer	Copies information; reads from the source and writes into the destination	Source or destination or both could be memory
2	Arithmetic	Performs required ALU operation and sets condition codes and flags	Operands have to be brought into ALU if not already in ALU
3	Logical	Same as above	Same as above
4	Program control	Program counter is modified	Branching
5	I/O	Inputs or outputs data; conducts I/O read or I/O write bus cycles	Performs data Transfer

Table 3.2 relates the instruction types and CPU actions. The processor executes a program by performing instruction cycles as shown in Fig. 3.15. Each instruction cycle consists of multiple steps as shown in Fig. 3.16. The first two steps (instruction fetch and instruction decode) are necessary for all instructions. The presence or absence of each of the remaining steps vary with instructions. This is well-explained in Tables 3.3, 3.4 and 3.5 which list the actions required for some common instructions.

The meaning of some of the instructions used in these tables is given below:

SKIP: Skip the current instruction

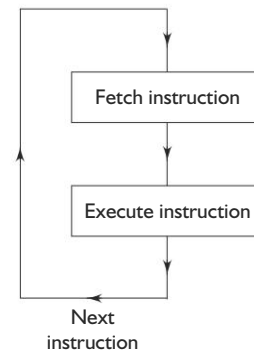
SKIP positive: Skip the current instruction only if the accumulator contents is a positive number

BUN: Branch unconditionally; same as JUMP

BZ: Branch if 'ZERO' flag is set

LDA: Load Accumulator

STA: Store Accumulator


Fig. 3.15 Instruction cycle

The overall role of the control unit is summarized in the following points:



IF : Instruction fetch

ID : Instruction decode

OAC : Operand address calculation

OF : Operand fetch

EO : Execute operation

SR : Store result

Fig. 3.16 Steps in instruction cycle

1. Fetch instruction.
2. Decode (analyze) opcode and addressing mode.
3. Generate necessary control signals corresponding to the instruction (opcode and addressing mode) in proper time sequence so that relevant microoperations are done.
4. Go to Step 1 for next instruction.

TABLE 3.3 Steps required for ADD, NOOP, HALT and SKIP instructions

Step No.	ADD	NOOP	HALT	SKIP
1	Instruction fetch	Instruction fetch	Instruction fetch	Instruction fetch
2	Instruction decode	Instruction decode	Instruction decode	Instruction decode
3	Operand address calculation if required	Go to next cycle	1. Reset RUN flip-flop 2. Go to next cycle	1. Increment program counter 2. Go to next cycle
4	Operand fetch	–	–	–
5	Execute operation (addition)	–	–	–
6	1. Store result 2. Go to next cycle	–	–	–

TABLE 3.4 Steps required for SKIPIFP, BUN, BZ and BRAS instructions

Step No.	SKIP positive	BUN	BZ	Branch and save
1	Instruction fetch	Instruction fetch	Instruction fetch	Instruction fetch
2	Instruction decode	Instruction decode	Instruction decode	Instruction decode
3	1. If sign is zero, increment the program counter 2. Go to next cycle	1. Enter branch address into program counter 2. Go to next cycle	1. If accumulator is zero, enter the branch address into program counter 2. Go to next cycle	Store program counter in the operand address location in memory
4	–	–	–	Load operand address into program counter
5	–	–	–	1. Increment program counter 2. Go to next cycle

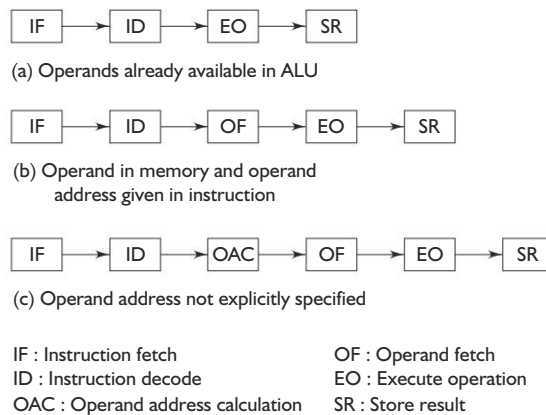
TABLE 3.5 Steps required for LDA, STA and AND instructions

Step No.	LDA	STA	AND
1	Instruction fetch	Instruction fetch	Instruction fetch
2	Instruction decode	Instruction decode	Instruction decode
3	Fetch operand from memory	Store accumulator contents in memory location	Fetch operands
4	1. Load operand in accumulator 2. Go to next cycle	Go to next cycle	Perform operation (AND)
5	–	–	1. Store result 2. Go to next cycle

The addressing mode affects how fast an instruction cycle is completed. Figure 3.17 shows three different cases for ADD instruction.

Apart from regular instruction cycle, the control unit also performs certain special sequences, listed as follows:

1. *Reset sequence* on (sensing reset signal).
2. *Interrupt recognition* (and branching to interrupt service routine).

**Fig. 3.17** Variations in ADD instruction cycle

3. *Abnormal condition recognition* (and taking appropriate action like shutdown or machine check handling).



3.8 Hardware–Software Interface

An object program can be written only if the programmer knows the computer's architecture clearly. As defined in section 1.18, a computer's architecture provides various attributes of the computer system to be known to a machine language programmer (or by a system software) to create a correct object program. At a higher level, the CPU executes the instructions in the program. The CPU performs (acts) according to the instructions in the program. The *status flags* and *control flags* provide a means of communication between the hardware and software.

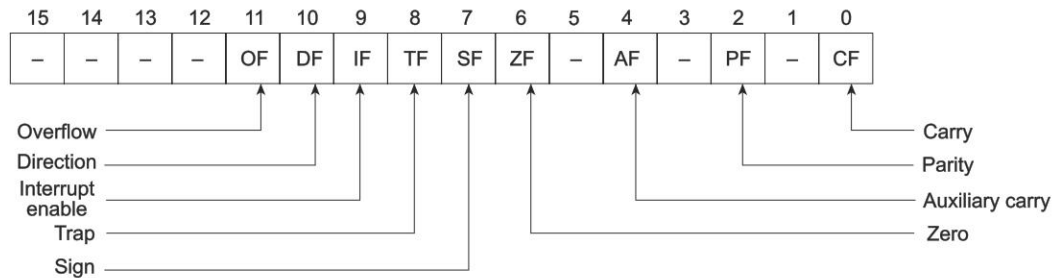
3.8.1 Status Flags

During the course of executing an instruction, the CPU sets certain information in a form that can be sensed by the software. For example, the INTEL 8086/8088 processor has status flags (Fig. 3.18) to register the status of the latest arithmetic or logical operation in ALU. By sensing the condition of these flags, the software understands the exact status of the CPU at any given time. To facilitate this, there are 'status flags' in CPU such as OVERFLOW flag, CARRY flag, SIGN flag etc. When an ADD instruction is executed, OVERFLOW and CARRY flags may be set by the CPU if corresponding condition is present in the ALU at the time of completion of the ADD instruction. To know the exact condition of the CPU, the program can sense the status of the flags by appropriate instruction immediately after the ADD instruction. In some CPUs, there are 'condition codes' that are used, by the CPU and the program, as means of communication about the condition in the CPU. The condition codes are created (by the CPU) on completion of any instruction. An appropriate instruction can be used as the next instruction to sense these condition codes. Alternatively, a conditional branch instruction can be used to sense and transfer control to another place in the program.

3.8.2 Control Flags

The control flags are used by the software to control certain aspects of processor operation. The control flags and status flags together is known as Flags Register or PSW (Program Status Word). Fig. 3.18 shows the control flags of INTEL 8086/8088 processor.

TRAP FLAG: This flag informs the processor to work in *single-step* mode. After each instruction, the processor creates an internal interrupt. This facility is generally used for *debugging* a newly written program to catch errors.

**Fig. 3.18** Flag Register

INTERRUPT FLAG: This flag informs the processor to honor the hardware interrupt if present.

DIRECTION FLAG: This flag is used for string manipulation. Depending on TRUE or FALSE, the *string processing* will be done from a higher address to lower address or from lower address to higher address.

3.8.3 Control Transfer Instructions

The *control transfer instructions* have special function of controlling processor operation. Table 3.6 lists some control transfer instructions and their functions.

TABLE 3.6 Control Transfer Instructions

No.	Instruction Name	Function
1	HALT	Stops instruction cycle by setting the HALT flag in the CPU.
2	SET CARRY	Sets the 'carry' flag
3	CLEAR CARRY	Resets the 'carry' flag
4	COMPLEMENT CARRY	Complements 'carry' flag
4	ENABLE INTERRUPT	Sets interrupt enable flag so that the CPU senses interrupts
5	DISABLE INTERRUPT	Resets interrupt enable flag so that the CPU ignores interrupts
6	ESCAPE	Indicates a special interpretation/action needed for the next instruction (as per the design/protocol)

 3.9 CISC vs RISC

There are two popular concepts associated with CPU design and instruction set:

- Complex Instruction Set Computing (CISC)
- Reduced Instruction Set Computing (RISC)

All relatively older systems (main frame, mini or micro) are CISC systems. Though today's systems comprise both types, RISC systems are more popular today due to their higher performance level as compared to CISC systems. However, due to high cost, RISC systems are used only when a special need for speed, reliability, etc. arises.

3.9.1 CISC TRENDS

From the early days, one of the parameters for computer evaluation is the instruction set. There are two critical aspects: total number of instructions, and their capabilities. Generally, the instruction set of a CISC system is made efficient by incorporating a large number of powerful instructions. The goal is to reduce the size of the compiled program (machine language) with limited number of instructions. Basically, a powerful instruction is equivalent to three or four simple instructions put together. Because of the limited size of the compiled program, the requirement for the main memory space is also small. In olden days, the main memory was based on magnetic core memory that was costly. Thus, including powerful instructions in the instruction set of a CPU reduces the main memory size and the cost. Another advantage of having powerful (complex) instructions is, the lesser the number of instructions in a (compiled) program, the lesser is the time spent by the CPU for fetching instructions. This decreases execution time considerably since the core memory (the olden day memory) is slow due to access time of 1 to 2 microseconds. Hence, we have dual benefits of powerful instructions in the instruction set: reduced system price and reduced program execution time. However, a highly efficient compiler is required to use the powerful instructions more frequently while translating the high level language program into a machine language program. Hence, the system software (compiler) becomes huge in order to generate a small object code. This does not affect the users as the compilation of an application program is a one time affair. Figure 3.19 illustrates the CISC scenario. Currently, computers use semiconductor memory as the main memory (and cache memory). It is cheaper and faster. Hence, the points discussed in favor of complex instruction set are not relevant in present times. However even today some computer professionals prefer CISC CPU due to their powerful instruction sets.

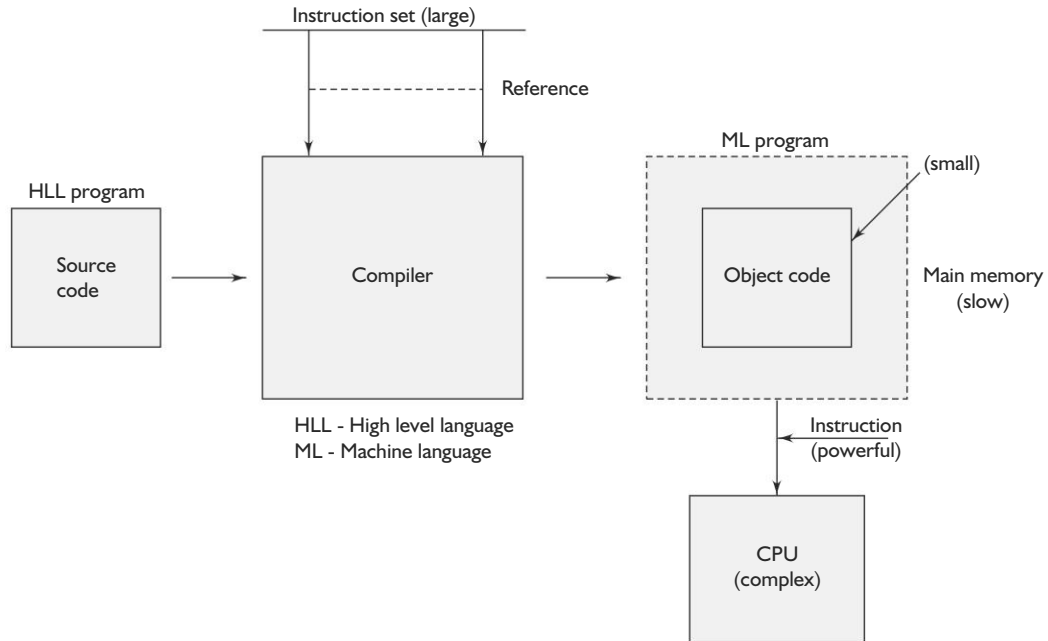


Fig. 3.19 CISC scenario

3.9.2 CISC Drawbacks

The CISC systems have the following drawbacks:

1. **CPU complexity**: The control unit design (mainly instruction decoding) is complex since the instruction set is large with heavily encoded instructions.
2. **System size and cost**: There is a lot of hardware circuitry due to complexity of the CPU. This increases the hardware cost of the system and also the power supply requirement.
3. **Clock speed**: Due to increased circuits, the propagation delays are more and the CPU cycle time is large and hence, the effective clock speed is reduced.
4. **Reliability**: The heavy hardware is prone to frequent failures.
5. **Maintainability**: Troubleshooting and detecting a fault is a big task, since there are a large number of complex circuits. The invention of microprogramming has reduced this burden to some extent. In-built diagnostic *microcodes* were provided in many CISC systems, giving a helping hand to the hardware engineer for handling system failures.

3.9.3 RISC Concept

The term 'KISS' is often used for RISC concept meaning 'Keep it short and simple'. The layman concept of RISC is this: 'being simple' helps doing 'more simple things' easily and reliably with limited resources and efforts, which in turn results in better efficiency. Technically speaking, the CPU's instruction set should have only few instructions, that too, simple instructions. Figure 3.20 depicts the RISC scenario.

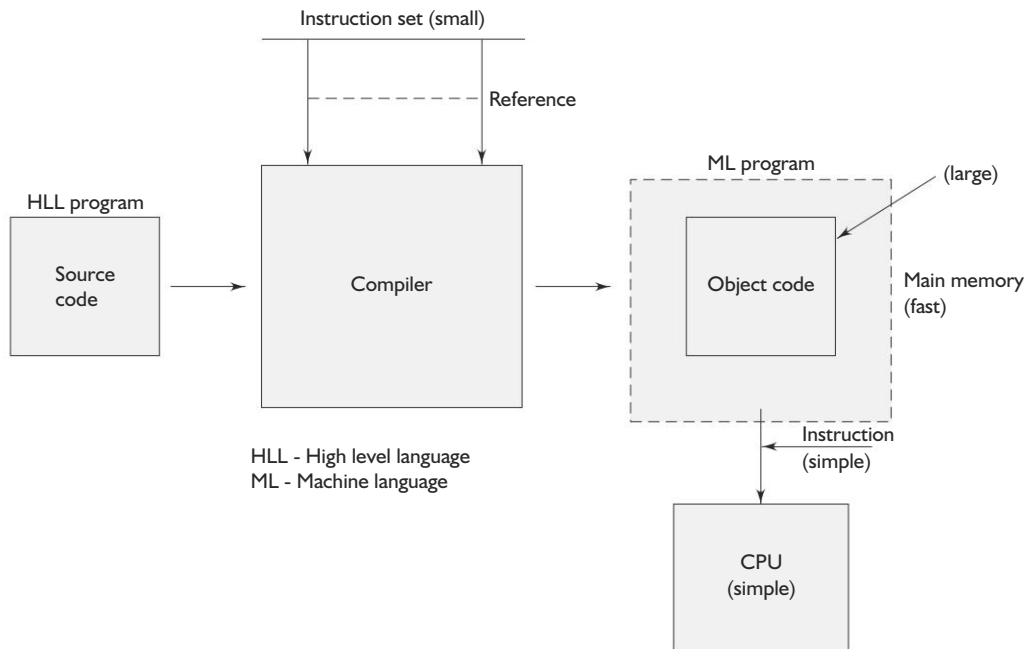


Fig. 3.20 RISC scenario

The RISC architecture has the following features:

1. Only simple instructions
2. Small instruction set
3. Equal instruction length for all instructions
4. Large number of registers for storing operands
5. Load/store architecture: The operand for an arithmetic instruction such as 'ADD' is available in a register. Similarly the result of an 'ADD' instruction is stored in a register and not in memory. Accordingly 'LOAD' instruction should precede an 'ADD' instruction and 'STORE' instruction should follow 'ADD' instruction. Hence, the compiler will generate a lot of 'LOAD' and 'STORE' instructions.

6. Faster instruction execution at a rate of one instruction per clock cycle takes place. Instruction pipelining, built-in cache memory and superscalar architecture are included in the CPU design so that on an average, one instruction comes out of the pipeline for every clock.

Chapter 13 covers RISC architecture in detail.

3.9.4 Sample RISC CPUs

Both microprocessor and non-microprocessor based RISC CPUs have been designed. Some of the RISC CPUs are as follows:

1. IBM RS/6000 or POWER architecture
2. Sun's SPARC family
3. HP's PA (Precision Architecture)
4. Motorola 88000 family
5. Intel 860
6. MIPS series
7. PowerPC



3.10 Instruction Set Design

The most significant and complex task in designing a computer is framing its instruction set. For achieving this goal, the following queries must be answered. How many instructions are needed? Which type of instructions have to be included in the instruction set? The early computers had unplanned instruction set. Their weak instruction set design drastically wasted the main memory space by lengthy (machine language) programs. Hence, a well-designed instruction set enables the compilers to create a compact object code (machine language program) saving upon the memory space.

A computer architect has to consider the following aspects before finalizing the instruction set:

1. *Programming convenience*: Number of instructions; programmers prefer to have as many instructions as possible so that appropriate operations are carried out by the respective instructions. But too many instructions in the instruction set results in a complex control unit design. Instruction decoding requires huge circuitry and time.
2. *Powerful addressing*: Programmers are happy if all possible modes of addressing the operands are present in the architecture. This gives a lot of flexibility to the programmer. But the control unit design becomes complex.

3. *Number of general purpose registers (GPRs)*: If the CPU has a large number of registers, the programmer finds data movement and processing faster. But, cost of CPU hardware increases with the large number of GPRs.
4. *Target market segment*: The application area may require certain special operations for efficient processing. A scientific computer should have floating-point arithmetic without which the precision would be heavily degraded. Whereas, a business computer must support decimal arithmetic, an entertainment computer must have multimedia operations.
5. *System performance*: If a program has less number of instructions, the performance of the system is enhanced since, time spent by the CPU in instruction fetching is reduced. To get a short program, instructions used should be powerful. Thus, a single instruction must be able to perform several micro-operations. Programmers appreciate this as it reduces the program size. But, the flipside of the coin is increase in both control unit complexity and instruction execution time. The modern concept of RISC architecture does not support complex instructions, though all the old CISC based computers have powerful instructions.

Traditionally, the superiority of a computer is decided on the basis of its instruction set. The total number of instructions and their powerfulness is given utmost importance since, these two factors contributed to the efficiency of the computer. An efficient program is the one which is short and occupies less memory space. Execution time is also a key factor.

The modern trend is use of simple instructions which results in a simplified control unit. The CPU speed is enhanced in RISC architecture.

3.10.1 Classification of Instruction Set Architectures

The selection of an instruction set for a computer depends on the manner in which the CPU is organized. Traditionally, there are three different CPU organizations with certain specific instructions:

1. Accumulator based CPU
2. Registers based CPU
3. Stack based CPU

3.10.1.1 Accumulator based CPU

Initially, most computers had accumulator based CPUs. It is a simple CPU in which the accumulator holds an operand for the instruction. Similarly, the instruction leaves the result

in the accumulator. The contents of the accumulator participate in the arithmetic operations such as addition, subtraction etc. This is also known as one address machine. The PDP-8, is the first minicomputer, which contains this type of CPU and is used for process control and laboratory applications. The Mark I computer cited in Chapter 2 is also a typical accumulator based computer. This style of CPU organization has become obsolete and replaced by the register based CPU.

Example 3.1 Write an assembly language program to derive the expression $X = (A + B) - (C + D)$ for an accumulator based CPU.

LOAD A — Loads accumulator with A
ADD B — Adds B to accumulator contents and stores the result in accumulator;
Accumulator contains $A + B$
STORE T — Stores $A + B$ in T, a temporary memory location
LOAD C — Loads C into accumulator
ADD D — Adds D to the accumulator contents and stores the result in
Accumulator; accumulator contains $C + D$
SUB T — Subtracts accumulator contents from T and stores the result in
accumulator
STORE X — Stores the accumulator contents in memory location X

The **advantages** of accumulator based CPU are as follows:

1. The accumulator contains an operand. Hence, there is no operand address field (for one operand) in the instruction. This results in short instructions and less memory space. Due to the absence of the address field, these are known as zero address instructions. Such CPUs normally have two types of instructions: zero address and single address. The single address instructions have one operand in main memory and the other in the accumulator.
2. Instruction cycle takes less time due to the absence of operand fetch.

The **disadvantages** of accumulator based CPU are as follows:

1. Program size increases due to several instructions. Hence the memory size increases.
2. Program execution time increases due to increase in the number of instructions.

3.10.1.2 Registers based CPU

In this type of CPU, multiple registers are used as accumulator. In other words, there are multiple accumulators. Such a CPU has a GPR organization. The use of the registers result in short programs with limited instructions. IBM System/360 and PDP-11 are some of the typical examples.

Example 3.2 Write an assembly language program to derive the expression $X = (A + B) - (C + D)$ for a registers based CPU.

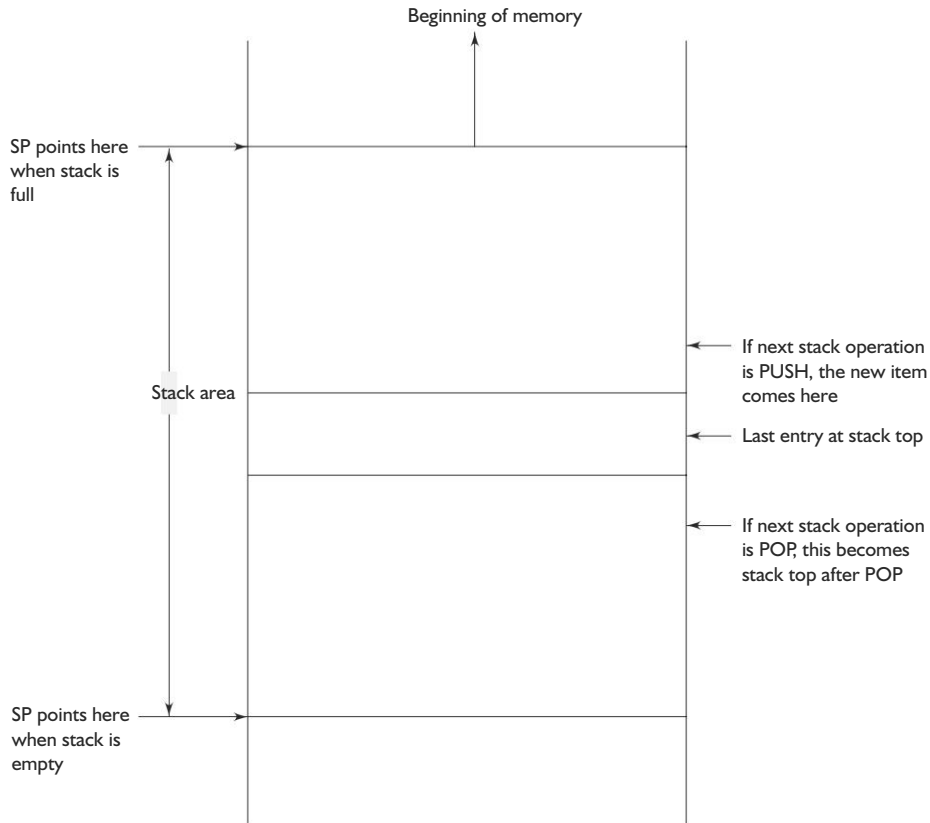
LOAD R1, A	—	Loads A into register R1
ADD R1, B	—	Adds B to R1 contents and stores the result in R1
LOAD R2, C	—	Loads C into R2
ADD R2, D	—	Adds D to R2 contents and stores the result in R2
SUB R1, R2	—	Subtracts R2 contents from R1 contents and stores the result in R1
STORE R1, X	—	Stores the result in memory location X

Comparing with example 3.1, it is observed that the registers based CPU (GPR architecture) results in shorter program size than the accumulator based CPU. Also, the program for the accumulator based CPU requires a memory location for storing partial result. Hence, additional memory accesses are needed during program execution. Thus, increase in the number of registers, increases CPU efficiency. But care should be taken to avoid unnecessary usage of registers. Hence, compilers need to be more efficient in this aspect.

3.10.1.3 Stack based CPU

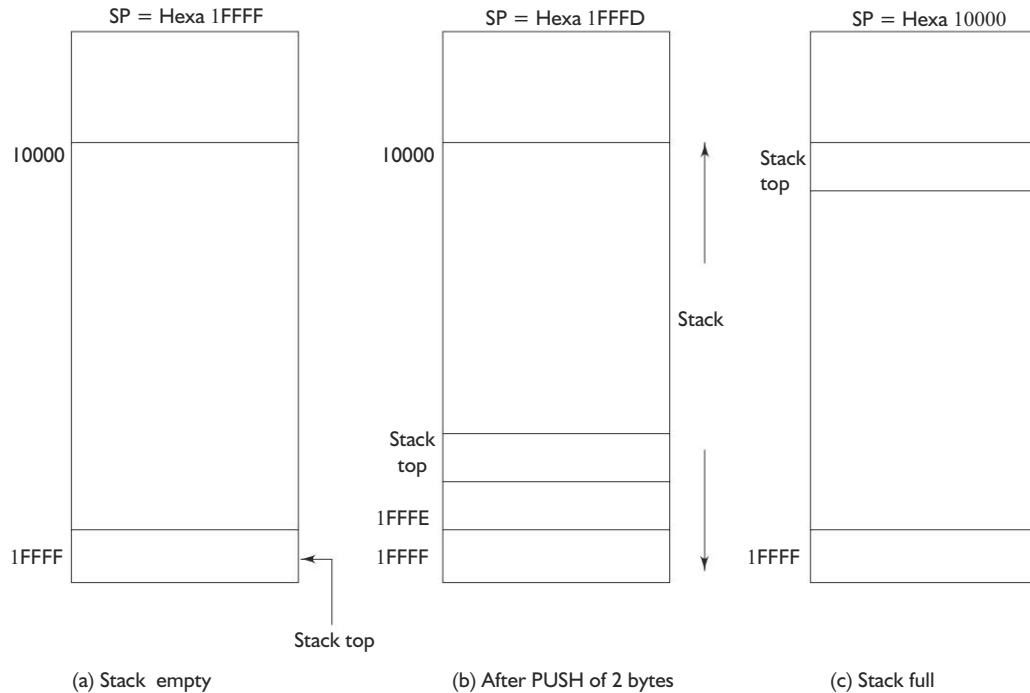
The stack is a push down list with a Last In First Out (LIFO) access mechanism. It stores the operands. It is present either inside¹ the CPU or a portion of the memory can be used as a stack. A register (or memory location) is used to point to the address of the top vacant location of the stack. This register is known as the Stack Pointer (SP). When nothing is stored in the stack, the stack is empty and the stack pointer points to the bottom of the stack. When an item is stored in the stack, it is called PUSH operation; the stack pointer is decremented. When the stack is full, the stack pointer points to the top of the stack. When any item is removed from the stack (POP operation), the stack pointer is incremented. The item which is pushed into the stack last (recently) comes out first in the next POP operation. In a stack based CPU, all operations by the CPU are done on the contents of a stack. Similarly, the result of an operation is stored in stack. Figures 3.21 and 3.22 illustrate the stack concept and mechanism. On executing an arithmetic instruction such as ADD, the top operands are popped-off the stack. Burroughs B5000 and HP 3000 are some examples of stack computers.

¹Implimenting the stack inside the CPU is a old technique which is no longer followed.

**Fig. 3.21** Stack concept

Example 3.3 Write an assembly language program to evaluate the expression $X = (A + B) - (C + D)$ for a stack based computer.

Statement	Stack contents after instruction execution	Stack locations occupied
PUSH A	A	1
PUSH B	A, B	2
ADD	A + B	1
PUSH C	(A + B), C	2
PUSH D	(A + B), C, D	3
ADD	(A + B), (C + D)	2
SUB	(A + B) - (C + D)	1
POP X	Empty (Nil)	0

**Fig. 3.22** Stack operation

From the contents of stack, it is observed that the stack grows when some PUSH operation takes place. When an arithmetic instruction is executed, the operands are removed from the stack and the result occupies a position in the stack top. Example 3.3 shows that the program size for the stack computer is more than for the registers based CPU.

The **advantages** of the stack based CPU:

1. Easy programming/high compiler efficiency
2. Highly suited for *block-structured languages*
3. Instructions don't have address field; short instructions

The **disadvantages** of the stack based CPU:

1. Additional hardware circuitry needed for stack implementation
2. Increased program size

3.10.2 Instruction Length

If the instruction is too long, it has following drawbacks:

1. The instructions occupy more memory space; this increases the memory requirement.
2. Either the data bus width has to be large or instruction fetch will take more time. The first condition increases hardware cost whereas, the other increases the instruction cycle time.

If the instruction is too short, it has following drawbacks:

1. There are too many instructions in the program. Hence, a lot of time is spent in fetching them.
2. Program size increases. Hence memory requirement increases.

3.10.3 Instruction Format

An instruction should provide four different information to CPU:

1. *Operation* to be done by the instruction.
2. *Operands* (data) on which operation has to be done.

ADD opcode	I operand	II operand	Result address	Next instruction address
------------	-----------	------------	----------------	--------------------------

Fig. 3.23 Four operand address instruction

3. *Location* (memory or register) where the result of the operation has to be stored.
4. Memory location from where the next instruction has to be fetched.

A theoretical instruction format specifying all the four items for an ADD instruction is shown in Fig. 3.23. In practice, some variations to this theoretical format are used as follows:

1. *Operand specification*: Instead of giving the operand in the instruction, its location in main memory is identified in the instruction as operand address.

This has two advantages:

- (a) The length of the operand address is shorter than the operand. Hence, it saves space in instruction length.
- (b) The programmer has flexibility in locating the operand in any of the following: main memory, CPU registers, instruction, I/O port, etc.

2. *Result address*: Instead of storing the result in a separate location, it is generally stored in the first or second operand address. This saves space in instruction length. The only disadvantage is that the initial operand is replaced by the result. The programmer should be careful in handling this problem. If the operand is needed by the program for future instructions, its copy should be retained in some other location so that it is not lost.

3. *Next instruction address*: In majority of the cases, the next instruction required is physically the next one after the current instruction. This implies that dedicating a field for the next instruction address is wastage of space and inefficient use of instruction length. Hence, it is assumed that the next required instruction is the physically next instruction. However, a branch (jump) instruction is used wherever necessary. The branch instruction specifies the address of the next instruction.

The format of ADD instruction shown in Fig. 3.24 incorporates the above variations. Some computers store the result in the first operand location whereas the others in the second location. The control unit is designed accordingly.

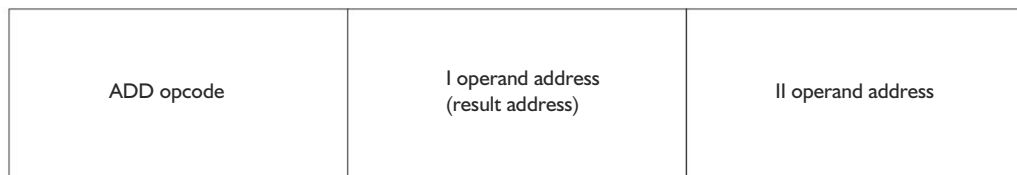


Fig. 3.24 Common format for ADD instruction

Example 3.4 The instruction length and operand address field is 36 bits and 14 bits, respectively. If two-operand instructions of 240 numbers are used, then how many one-operand instructions are possible?

Instruction length = 36 bits

A two-operand instruction needs 28 bits for the operand addresses (2×14). Hence, opcode size = $36 - 28 = 8$ bits

Total number of instructions possible = 256

Number of one-address instructions = $256 - 240 = 16$

3.10.4 Location of Operands

There are multiple options for placing the operands: main memory, CPU register, I/O port, and instruction itself. Keeping an operand in a CPU register is more effective than storing in the main memory due to shorter access time of CPU registers. This results in the reduction of instruction cycle time. Locating an operand in the instruction is used for only certain

special instructions. The content of a port can be used as an operand just like the content of a memory location. There are some instructions which have no operands. HALT and NOOP are typical examples. There are some instructions which test the status of hardware components such as register, flip-flop, memory location, etc. In these cases, there are no operands. Similarly, some instructions just wait for some external signal.

3.10.5 Location of Result

There are multiple places for storing the result: main memory, CPU register, output port etc. Some instructions such as HALT and NOOP do not have explicit results whereas, some instructions just set or reset some flip-flops or clear registers.

3.10.6 Variable Length Instructions

If all instructions have equal length, it causes wastage of memory space. This is because some simple instructions such as HALT, NOOP, etc. have only opcodes and hence, the remaining space in the instructions is unused. It also leads to wastage of time in fetching the entire instruction. Hence, in practice, a computer's instructions do not have equal length.² Some will be short (one byte, two bytes) and others will be long (three bytes and above). The opcode of the instruction usually gives the information about the length of the instruction. By looking at the opcode, the CPU decides about the additional memory read operations required to fetch the full instruction.

The programmers like to have different instruction lengths as it gives them flexibility and also save the memory space. However, there is a drawback in this. The CPU design becomes complex. It should be capable of expanding and contracting the instruction fetch sequence based on the opcode, as shown in Fig. 3.25.

3.10.7 Data Ordering and Addressing Standards

There are two different conventions followed for storing information in memory and addressing: *Big-endian* assignment and *Little-endian* assignment. Suppose we have a 32-bit information 342CE11A (hexa) to be stored in memory from location 1000 onwards. Since there are 4 bytes, the information occupy addresses 1000 to 1003. In big-endian assignment, the most significant byte is stored in lower address and least significant byte is stored in

²RISC CPUs are exceptions. All the instructions for a RISC CPU are equal in size.

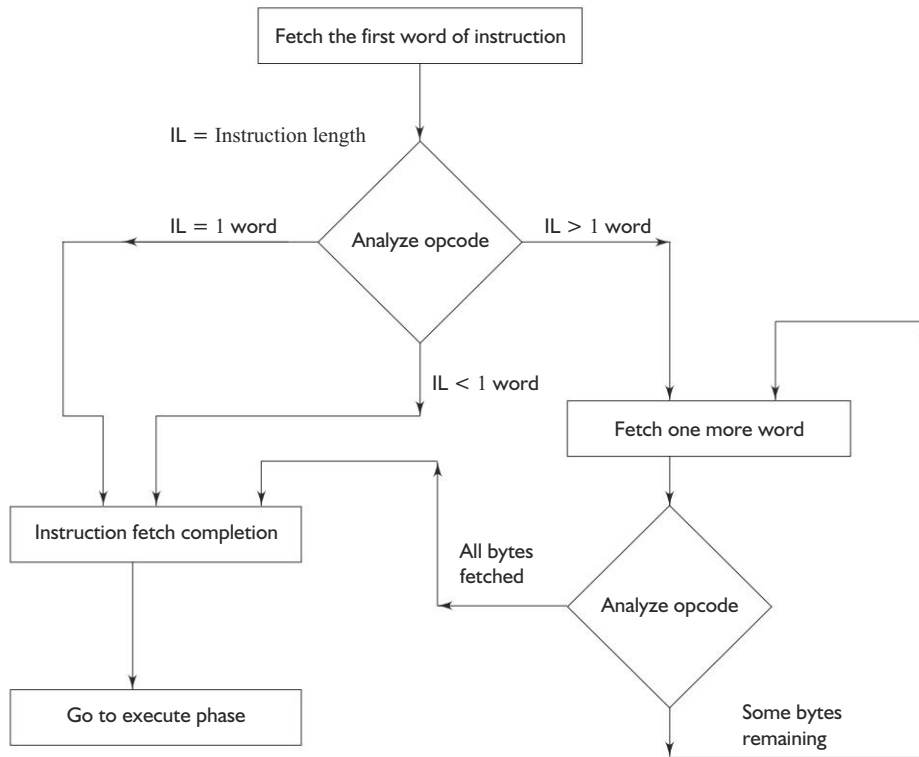


Fig. 3.25 Instruction fetch sequence for variable length instruction CPU

higher address. In little-endian assignment, the least significant byte is stored in lower address and the most significant byte is stored in higher address. Accordingly, the byte arrangements and memory addresses are as follows:

- Address and Data Storage—big-endian: 1000 (34), 1001 (2C), 1002 (E1), 1003 (1A)
- Address and Data Storage—little-endian: 1000 (1A), 1001 (E1), 1002 (2C), 1003 (34)

1000	1001	1002	1003
34	2C	E1	1A

Big-endian

1000	1001	1002	1003
1A	E1	2C	34

Little-endian

In case of 64-bit data (8 bytes), the remaining 4 bytes are continued from address 1004 onwards applying the same rules for the second word of 4 bytes. Some computers can handle only one standard where as some could handle both.

3.11 Instruction Types and Operations

The instructions are classified into different types on the basis of the following factors:

1. *Opcode*: Nature of operation done by the instruction.
2. *Data*: Type of data: binary, decimal etc.
3. *Operand location*: Memory, register etc.
4. *Operand addressing*: Method of specifying the operand location (address).
5. *Instruction length*: One byte, two byte etc.
6. *Number of address fields*: Zero address, single address, two address etc.

No two computers of different models have same instruction set. Almost every computer has some unique instructions which attract the programmers. Computer architects give considerable attention to the framing of the instruction set since it affects both the programmer and the computer machine. Taking into account the various operations, the instructions can be classified into following eight types:

1. *Data transfer instructions*: These move the data from one register/memory location to another.
2. *Arithmetic instructions*: These perform arithmetical operations.
3. *Logical instructions*: These perform Boolean logical operations.
4. *Control transfer instructions*: These modify the program execution sequence
5. *Input/output (I/O) instructions*: These transfer information between external peripherals and system nucleus (CPU/memory).
6. *String manipulation instructions*: These manipulate strings of byte, word, double word etc.
7. *Translate instructions*: These convert the data from one format to another.
8. *Processor control instructions*: These control processor operation.

Table 3.7 lists some sample instructions for each type. Since, different computer manufacturers follow different instruction notations, a simple mnemonics is adopted in this table for better comprehension.

Table 3.7 Instruction types examples

Type no.	Type	Specific instructions	
		Name	Action
1	Data transfer	MOVE	Transfer data from the source location to destination location
		LOAD	Transfer data from a memory location to a CPU register
		STORE	Transfer data from a CPU register to a memory location
		PUSH	Transfer data from the source to stack (top)
		POP	Transfer data from stack (top) to the destination
		XCHG	Exchange; swap the contents of the source and destination
		CLEAR	Reset the destination with all '0'
		SET	Set the destination with all '1'
2	Arithmetic	ADD	Add; calculate sum of two operands
		ADC	Add with carry; calculate sum of two operands and the 'carry' bit
		SUB	Subtract; calculate the difference of two numbers
		SUB B	Subtract with borrow; calculate the difference with 'borrow'
		MUL	Multiply; calculate product of two operands
		DIV	Divide; calculate quotient and remainder of two numbers
		NEG	Negate; change sign of operand
		INC	Increment; add 1 to operand
		DEC	Decrement; subtract 1 from operand
		SHIFT A	Shift arithmetic; shift the operand (left or right) with sign extension
3	Logical	NOT	Complement the operand
		OR	Perform bit-wise logical OR of operand
		AND	Perform bit-wise logical AND of operands
		XOR	Perform bit-wise 'exclusive OR' of operands
		SHIFT	Shift the operand (left or right) filling the empty bits positions as 0's

(Contd.)

Type No.	Type	Specific instructions	
		Name	Action
4	Control transfer	ROT	Rotate; shift operand (left or right) with wrap-around
		TEST	Test for specified condition and set or reset relevant flags
		JUMP	Branch; enter the specified address into PC; unconditional transfer
		JUMPIF	Branch on condition; enter the specified address into PC only if the specified condition is satisfied; conditional transfer
		JUMPSUB	CALL; save current 'program control status' (into stack) and then enter specified address into PC
		RET	RETURN; unsave (restore) 'program control status' (from stack) into PC and other relevant registers/flags
		INT	Interrupt; creates a software interrupt; saves 'program control status' (into stack) and enter the address corresponding to the specified (vector) code into PC
5	Input-output instructions	IRET	Interrupt return; restore (unsave) 'program control status' (from stack) into PC and other relevant registers and flags
		LOOP	Iteration; decrement the implied register by 1 and test for non-zero; if satisfied, enter the specified address into PC
		IN	Input; read data from specified input port/device into specified or implied register
		OUT	Output; write data from specified or implied register into an output port/device
		TEST I/O	Read the status from I/O subsystem and set condition flags (codes)
6	String manipulation	START I/O	Inform the I/O processor (or data channel) to start the I/O program (commands for the I/O program)
		HALT I/O	Inform the I/O processor (or data channel) to abort the I/O program (commands for the I/O operations) under progress
		MOVS	Move byte or word of string
		LODS	Load byte or word of string
		CMPS	Compare byte or word of strings
		STOS	Store byte or word of string
		SCAS	Scan byte or word of string

(Contd.)

Type No.	Type	Specific instructions	
		Name	Action
7	Translate	XLAT	Translate; convert the given code into another by table lookup
		PACK	Convert the unpacked decimal number into packed decimal
		UNPACK	Convert the packed decimal number into unpacked decimal
8	Transfer control	HLT	Halt; stop instruction cycle (processing)
		STI (EI)	Set interrupt (enable interrupt); sets interrupt enable flag to '1' (so as to allow maskable interrupts)
		CLI (DI)	Clear interrupt (disable interrupt); resets interrupt enable flag to '0' (so as to ignore maskable interrupts)
		WAIT	Freeze instruction cycle till a specified condition is satisfied (such as an input signal becoming active)
		NOOP	No operation: nothing
		ESC	Escape; the next instruction (after ESC) is for the coprocessor and not for the main CPU
		LOCK	Reserve the bus (and hence memory) till the next instruction (following LOCK instruction) is executed (completed)
		CMC	Complement 'carry' flag
		CLC	Clear 'carry' flag
		STC	Set 'carry' flag

3.11.1 Macros and Subroutines

Macros and subroutines are special mechanisms that eliminate repetitive programming. For performing a specific task at many places in a program (with different data at each place) using one of these mechanisms help avoid writing the task-routine many times.

3.11.1.1 Macro

Macro is a routine that can be invoked in any place in a program by just naming it. It is an independent subprogram with some parameter input whose value has to be supplied at the place of invoking the macro. A macro can be invoked at multiple places, if required. In the object code of the program, the code of the macro is inserted in every place where the macro is invoked. For instance, a square root calculation is required at many places in a

program. The square root calculating routine can be made as a macro named SQR. In the following example, the SQR macro is invoked in two places with different parameter values, 1000 and 2000.

SQR MACRO number

```

.....
.....
END MACRO
.....
.....
SQR 1000
.....
.....
SQR 2000
.....

```

3.11.1.2 Subroutine

The subroutine is a program to which the CPU temporarily branches (from the main program) to perform a special function. Before entering the subroutine, the main program places the parameters either in the registers or in the memory locations. Then the 'place' in the main program from where the branch is made to the subroutine is marked by storing the program counter value in memory (or CPU register). A CALL statement in the main program causes the branch to the subroutine. The subroutine retrieves the required parameters and performs the operations. Then, the subroutine (called program) transfers control to the main (calling) program. This is achieved by using a RETURN statement (as the last statement) in the subroutine. The following example illustrates the subroutine concept.

Main program

```

....
....
....
CALL SQR
....
....
SQR .....
....
....
RETURN

```

It is possible to have *subroutine calls* within the subroutine. This is called subroutine nesting. It is similar to the concept of nested interrupts.

 3.12 Addressing Modes

The two common places used for locating the operands of an instruction are the main memory and the CPU registers. If the operand is located in the main memory, the location address has to be given by the instruction in the operand field. It is not necessary to give the address explicitly in the instruction. Many methods are followed to specify the operand address. These are known as *addressing modes*. A given computer may not use all the addressing modes. The popular addressing modes are as follows:

1. Immediate addressing
2. Direct (absolute) addressing
3. Indirect addressing
4. Register addressing
5. Register indirect addressing
6. Relative addressing
7. Index addressing
8. Base with index addressing
9. Base with index and offset addressing

Why do we need so many addressing modes? Multiple addressing modes give flexibility to the programmer in writing efficient (short and fast) programs. The following goals influence the computer architect while choosing the addressing modes:

1. Reducing the instruction length by having a short field for the address.
2. Providing powerful aids to the programmer for complex data handling such as indexing of an array, loop control, program relocation, etc.

The exact addressing mode used by an instruction is indicated to the control unit, in any of the following two ways:

1. A separate mode field in the instruction indicates the addressing mode used, as shown in Fig. 3.26.

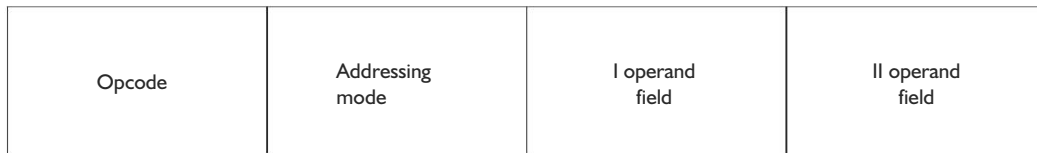


Fig. 3.26 Addressing mode field

2. The opcode itself explicitly specifies the addressing mode used in the instruction.

A given computer is designed to use one of the two techniques. Table 3.8 briefly defines the addressing modes. A detailed description of the addressing modes is given below:

TABLE 3.8 Addressing modes and mechanisms

S. no.	Addressing mode	Mechanism	Remarks
1	Immediate	Operand is present in the instruction itself	Fast operand fetch
2	Direct	Operand is in memory; its location address is given in the instruction	One memory access required for operand fetch
3	Indirect	Operand is in memory; its address is also in memory; address of the location containing operand address is given in the instruction	Additional memory access required for knowing the operand address
4	Register direct	Operand is in a register; the register address (number) is given in the instruction	Quickest access of operand without any memory access; faster than direct addressing
5	Register indirect	Operand is in memory; its address in a register; address (number) of the register containing the operand address is given in the instruction	Faster than indirect addressing
6	Relative addressing	The operand is in memory; its relative position (offset number) with respect to the contents of program counter is given in the instruction	Quick address calculation without memory access
7	Base register addressing	The operand is in memory; its address is specified in two parts; the instruction gives an offset number and also specifies the base register; the offset number is added to the base register contents	—
8	Index addressing	The operand is in memory; the instruction contains an address and the index register contains offset number; the address and the offset are added to get the operand address	—

3.12.1 Immediate Addressing (Fig. 3.27)



Fig. 3.27 Immediate addressing

This mode is a simple one with no operand fetch activity. The instruction itself contains the operand. The following examples give assembly language statements for the immediate addressing mode. The (#) sign is used to indicate that the constant following the sign is the immediate operand.

- **MOVE #26, R1** or **MVI R1, 26**—Loads binary equivalent of 26 in register R1
- **ADD #26, R1**—Adds binary equivalent of 26 to R1 and stores the result in R1
- **CMP #26, R1** or **CMI R1, 26**—Matches R1 contents with binary equivalent of 26

Merits: This operand is available in the CPU as soon as the instruction fetch is over. Hence the instruction cycle winds-off fast.

Demerits: The value of the operand is limited by the length of the operand field in the instruction.

3.12.2 Direct Addressing (Fig. 3.28)



<Memory address> = Operand

Fig. 3.28 Direct addressing

Since the operand address is explicitly given in the instruction, this mode is called direct addressing mode. The following assembly language statements illustrate this mode:

- **LOAD R1, X** — Loads the contents of memory location X in register R1
- **MOV Y, X** — Moves the contents of memory location X to the location Y. (Here both the operands are specified using direct addressing mode)
- **JUMP X** — Transfers program control to instruction at memory location X. (X is not operand address but branch address)

Merits: Since the operand address is directly available in the instruction, there is no need for the operand address calculation. Hence, instruction cycle time is reduced.

Demerits: The size of the operand address is limited by the operand field in the instruction.

3.12.3 Indirect Addressing (Fig. 3.29)

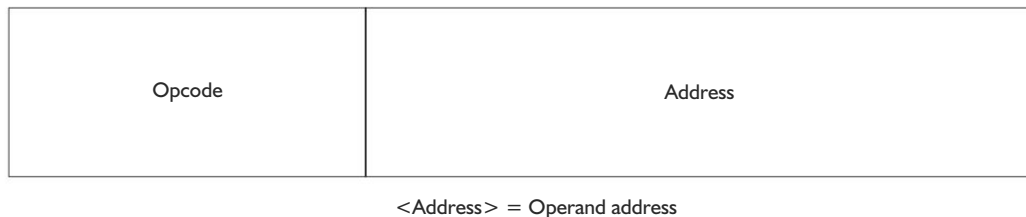


Fig. 3.29 Indirect addressing

The instruction gives an address of a location (X) which, in turn contains another location's address (Y), which actually contains the operand. It can be represented as

$$\langle X \rangle = Y, \langle Y \rangle = \text{Operand}$$

The Y is known as the pointer. The value of Y (address) can be changed dynamically in a program, without changing the instruction, by simply modifying the contents of location X. Multi-level indirect addressing is also possible. The following example illustrates the indirect addressing mode:

- **MOVE (X), R1** — Contents of the location whose address is given in X is loaded into register R1

Merits: Flexibility in programming; changing the address during program run-time without changing the instruction contents. It is useful for implementing pointers (C language).

Demerits: Instruction cycle time increases since two memory accesses are required for a single level indirect addressing.

3.12.4 Register (Direct) Addressing (Fig. 3.30)



<R> = Operand

Fig. 3.30 Register addressing

Conceptually, register addressing is similar to direct addressing except that instead of the memory location, the register holds the operand. The instruction contains the register number that has the operand. This mode is very useful to a long program in storing the intermediate results in the registers rather than in memory. The following examples illustrate the register addressing mode:

- **ADD R1, R2** — Adds the contents of registers R1 and R2 and stores the result in R1. Both the operands are addressed in register addressing mode
- **STORE R1, MEM1** — Contents of register R1 are stored in memory address MEM1; register addressing is used for first operand and direct addressing is used for second operand.

Merits: Faster operand fetch without memory access.

Demerits: Number of registers is limited and hence, effective utilization by the programmer is essential. Otherwise, program execution time will increase.

3.12.5 Register Indirect Addressing (Fig. 3.31)



<R> = Operand address

Fig. 3.31 Register indirect addressing

In this mode, a register is used to keep the memory address of the operand. Thus, the register acts as the memory address register. This mode is very useful for quick access of the main memory locations such as an array. The instruction with register indirect mode is part of a loop. Initially, the beginning address of the array is stored in the register. When the instruction is encountered for the first time, the first entry of the array is accessed. Then, the contents of the register is increased by 1, by another instruction in the loop (before the register indirect mode instruction). Thus, every time this instruction is executed the next entry in the array is accessed.

Merits: Effective utilization of the instruction length since specifying register number needs only few bits.

3.12.6 Relative Addressing (Fig. 3.32)



$$\text{Operand address} = \text{PC} + \text{Offset}$$

Fig. 3.32 Relative addressing

In this mode, the instruction specifies the operand address (memory location) as the relative position of the current instruction address, i.e., the contents of PC. Hence, the operand is situated at 'short distance' from the PC contents. Generally, this mode is used to specify the branch address in the branch instruction, provided the branch address is nearer to the instruction address. The following examples illustrate this mode.

JUMP + 8 (PC)

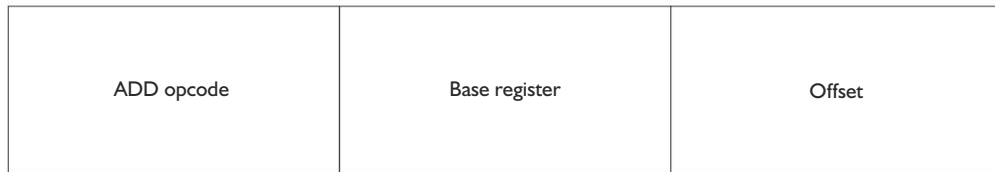
JUMP - 8 (PC)

Demerits: Smaller number of bits in the address field.

3.12.7 Base Register Addressing (Fig. 3.33)

This mode is used for relocation of the programs in the memory (from one area to another). In base register addressing mode, the instruction gives the displacement relative to the base address (contents of base register). Hence, to relocate the operand from the current memory area to another memory area, the base register is loaded with the new base address. The

instruction need not be modified. In this way, an entire program or a segment of program can be moved from one area of the memory to another without affecting the instructions, by simply changing the base register contents. This is useful for multiprogramming systems since for different times (runs), different area of the memory is allotted for a program. A CPU can have multiple base registers.



$$\text{Operand address} = \text{Base register} + \text{Offset}$$

Fig. 3.33 Base register addressing

Merits: The operand address field in the instruction is very short since it only gives offset (displacement); the operand address is calculated without memory access.

3.12.8 Index Addressing (Fig. 3.34)



$$\text{Operand address} = \text{Address} + \text{Index register}$$

Fig. 3.34 Index addressing

The index addressing mode is slightly different from the base register addressing mode. An index register contains an offset or displacement. The instruction contains the address that should be added to the offset in the index register, to get the effective operand address.

Generally, the address field gives the start address of an array in memory. The index register contains the 'index value' for the operand i.e. the difference between the start address and the operand address. By changing the value of the index register, any operand in

the array can be accessed. Usually, the operands (array elements) are in consecutive locations. They are accessed by simply incrementing the index register.

Some CPUs support 'autoindexing' features, which involve automatic incrementing (by hardware) of the index register whenever an instruction with index addressing is executed. This eliminates the use of a separate instruction to increment the content of the index register. This also results in faster action as well as shorter program size. However, the control unit has additional responsibility of 'autoindexing'.

3.12.9 Stack Addressing

In stack addressing mode, all the operands for an instruction are taken from the top of the stack. The instruction does not have any operand field. For example, an ADD instruction gives only the opcode (ADD). Both the operands are in the stack in consecutive locations. When the ADD instruction is executed, two operands are popped-off the stack one-by-one. After addition, the result is pushed onto the stack.

Merits: No operand fields in the instruction. Hence, the instruction is short.



3.13 Data Representation

Depending on the problem, different application programs use different types of data. A machine language program can operate either on numeric data or non-numeric data. The numeric data can either be binary or decimal number. The non-numeric data are of the following types:

1. Characters
2. Addresses
3. Logical data

All non-binary data is represented inside a computer in the binary coded form.

3.13.1 Character Data

A group of bits are used to represent a character which may be a digit, an alphabet or a special symbol etc. A string of multiple characters usually form a meaningful data. Some sample examples are as follows:

1. C#
2. ISO 9000
3. IBM System/360
4. Pentium 4
5. bgrajulu@gmail.com

In the past, several types of codes were used for representing the various characters. The popular MORSE code was used in telegraphy and teleprinters. It had a 5-bit code where each character is represented by a 5-bit pattern. The American National Standard Institute (ASCII) has announced the “American Standard Code for Information Interchange” (ASCII). This standard uses an 8-bit pattern in which 7 bits specify the character. The 8th bit is generally used as a parity bit for error (a bit drop or a bit pick up) detection. But, some computers do not use the eighth bit. Several computers follow the ASCII standard which is discussed in Annexure 3. Another popular code is Extended BCD Interchange Code (EBCDIC) used by some computer systems such as IBM System/360. It is an 8-bit code.

3.13.2 Addresses

For some instructions, the operand is an address. An operation (arithmetic or logical) may or may not be involved. Depending on the instruction, the address is treated as a binary or a logical number. In some instructions, the operand address is specified in multiple parts as discussed under addressing modes.



3.14 Binary Data

The binary data can be represented either as a fixed-point or a floating-point number. In fixed-point number representation, the position of the binary point is rigidly fixed in one place. In floating-point number representation, the binary point's position can be anywhere. The fixed-point numbers are known as *integers* whereas the floating-point numbers are known as *real numbers*. Arithmetic operations (addition etc.) on fixed-point numbers are simple and they require minimum hardware (circuits). The floating-point arithmetic is difficult and requires complex hardware. Compared to fixed-point numbers, the floating-point numbers have two advantages:

1. For any given number of data bits, the maximum or minimum value that can be represented in floating-point number representation is higher as compared to fixed-point number representation. It is useful in dealing with very large or very small numbers.
2. The floating-point number representation is accurate in arithmetic operations. Chapter 4 covers both fixed-point and floating-point number representations.

SUMMARY

Several parameters of CPU have direct impact on system performance and programmer productivity. The most significant and complex task in a computer design is framing an instruction set. Traditionally, a computer's superiority was decided based on the richness of its instruction set. The total number of instructions and its powerfulness are very important since these two factors contribute to the efficiency of programming the computer. The instruction set of a CISC system is made powerful by incorporating a large number of powerful instructions. A short and fast running program is desired.

The modern trend is choosing a simple instruction set for designing a simpler control unit. The RISC architecture is covered by the following features: simple instructions, small instruction set, equal instruction length for all instructions, large number of registers and LOAD/STORE architecture.

Many initial computers had accumulator based CPUs. Later computers had registers based CPU in which there are multiple registers each of which can be used as accumulator. Stack based CPUs are rare but significant.

The instructions can be classified into following eight types:

1. *Data transfer instructions*: these move data from one register/memory location to another.
2. *Arithmetic instructions*: these perform arithmetical operations.
3. *Logical instructions*: these perform Boolean logical operations.
4. *Control transfer instructions*: these modify the program execution sequence.
5. *Input/output (I/O) instructions*: these transfer information between external peripherals and system nucleus (CPU/memory).
6. *String manipulation instructions*: these manipulate strings of byte, word, double word etc.
7. *Translate instructions*: these convert data from one format to another.
8. *Processor control instructions*: these control processor operation.

The different modes for specifying the operand address in the instructions are known as addressing modes. A given computer may not use all the addressing modes. The popular addressing modes are as follows:

1. Immediate addressing
2. Direct (absolute) addressing
3. Indirect addressing
4. Register addressing
5. Register indirect addressing
6. Relative addressing
7. Index addressing
8. Base with index addressing
9. Base with index and offset addressing

The following goals influence the computer architect in making the choices for the addressing modes:

1. Reducing the instruction length by using a short field for the address.
2. Providing powerful utility to the programmer for complex data handling such as indexing of an array, loop control, program relocation etc.

A machine language program can operate on numeric or non-numeric data. The numeric data can be either a binary or a decimal number. The non-numeric data can be any of the following types:

1. Characters
2. Addresses
3. Logical data

All the non-binary data is represented inside a computer in binary coded form. The binary data can be represented either as a fixed-point number or floating-point number. Arithmetic operations (such as addition etc.) on fixed-point numbers are simple and requires minimum hardware (circuits), whereas the floating-point arithmetic is complex and needs elaborate hardware. Compared to fixed-point number, the floating-point number has two advantages:

1. For any given number of data bits, the maximum or minimum value that can be represented in floating-point number representation is more than that in fixed-point number representation. It is useful in dealing with very large or very small numbers.
2. The floating-point number representation is more accurate in arithmetic operations.

Functionally the processor is divided into datapath and control unit. The datapath plays an important role in the processor. Its organization has direct influence on the system

performance and programming efficiency. The processor can be viewed at four different levels: System level, Instruction set level or Architecture level, Register Transfer level (RTL) and Gate level.

The overall goal for designing a processor is to meet the needs of the instruction set. This involves designing the appropriate hardware so that all macro-operations can be executed accurately as required by the instructions. The designing of a processor is done systematically following a sequence of steps. The RTL is a notation used to specify the micro-operation transfer between the registers. It is a tool used for describing the behavior of instructions and organization of a computer.

The micro-operations are classified into four types.

1. Register transfer microoperations.
2. Arithmetic microoperations.
3. Logic microoperations.
4. Shift microoperations.

REVIEW QUESTIONS

1. The stack based CPUs don't have registers for storing the operands. Can we conclude that the stack based CPU has less hardware circuits and hence cheaper than registers-based CPU? (Hint: stack control mechanism).
2. Which type of CPU is smaller and hence cheaper—single accumulator based CPU or registers based CPU? Justify your answer.
3. A registers based CPU can be viewed as multiple accumulators based CPU. Justify this statement.

EXERCISES

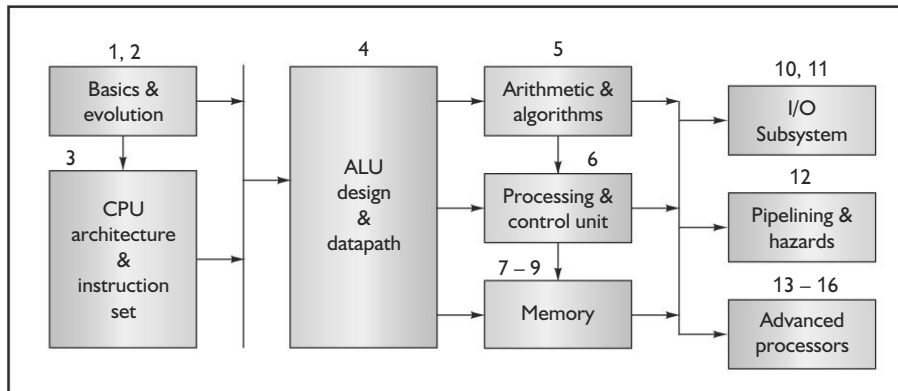
1. A program has three jump instructions in three consecutive words (locations) in memory: 0111, 1000 and 1001. The corresponding jump addresses are 1001, 0111, 0111 respectively. Suppose we load 0111 initially in PC and start the CPU, how many times will the instruction in location 1000 be fetched and executed?
2. Write a program to evaluate the expression $X = (A \times B) + (C \times D)$ for:
(a) accumulator based CPU; (b) registers based CPU and (c) stack based CPU.
3. The instruction format of a CPU is designed for following two types: (a) opcode and three fields for register addresses; (b) opcode and one field for memory address. Identify different formats for instruction.

4. A CPU's instruction set has 12-bit instructions. Four bits are allowed for each operand field. Three different types of instructions are supported: (a) m number of zero-address instructions; (b) n number of two-address instructions and (c) remaining number of one-address instructions. What is the maximum possible number of one-address instruction?
5. A 32-bit CPU has 16-bit instructions and 12-bit memory address. Its memory is organized as 4 k words of 32-bits each. Each memory word stores two instructions. The instruction format has 4-bits for opcode. The CPU's instruction register can accommodate two instructions. Suggest a design strategy for the instruction cycle sequence.

CHAPTER 4



ALU DESIGN AND DATAPATH



Chapter Flow Diagram

OBJECTIVES

- ◀◀ *Fixed-point numbers*
- ◀◀ *Floating-point numbers*
- ◀◀ *ALU design*
- ◀◀ *Datapath organization*

KEYWORDS

Combinational ALU

Sequential ALU

Normalization

Mantissa

Exponent

Excess code

Hidden 1 principle

Coprocessor

Fixed-point number

Floating-point number

4.1 Introduction

The datapath plays an important role. It contains the hardware paths and circuits that are necessary for moving the data and performing various operations. The speed of the datapath unit depends on both the algorithms used for the arithmetic operations and the design of the ALU circuits. This chapter covers the ALU circuits which are necessary for performing the arithmetic operations and associated datapath. The algorithms for the arithmetic operations are covered in Chapter 5.

4.2 Arithmetic Types

Theoretically, a computer can operate on any type of data through appropriate programming. A given computer is practically designed with a limited type of arithmetic. Figure 4.1 shows popular arithmetic types used in computers. Since the internal circuits of

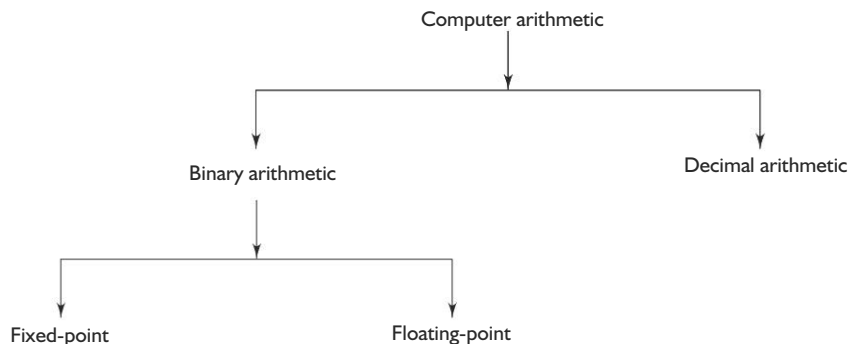


Fig. 4.1 Computer arithmetic types

computers are binary in nature, binary arithmetic is available in all the computers. The user uses decimal number system while all the input to computers are decimal. These are converted into binary numbers by the computer before performing the arithmetic operations. The final result is converted into decimals and presented to the user. This concept works well for limited number of conversions. In business applications, the amount of data is large but the extent of arithmetic operation is limited. In these cases, a lot of time is wasted in conversions if we use binary arithmetic. Hence computers are designed with decimal arithmetic for such applications. Decimal numbers are represented in Binary Coded Decimal (BCD). BCD arithmetic algorithms are used in decimal arithmetic. Decimal arithmetic has two disadvantages:

1. The hardware for decimal arithmetic is complex and expensive.
2. The BCD number system does not make efficient use of binary bit combinations—some amount of wastage is inevitable. Hence, utilization of storage hardware (memory and registers) is also poor thereby increasing hardware cost.

Fixed-point arithmetic is also known as integer arithmetic. In most of the computers, complementary arithmetic is followed for integer arithmetic. It involves sequences of addition operations on two's-complement values. This means the arithmetic can be done with an adder and an inverter (complementer).

Real numbers can contain both whole and fractional parts. Inside the computer, the real numbers are generally represented in floating-point notation. The position of radix point is not fixed. It can be moved anywhere within a bit string. Hence this notation offers flexibility of manipulation.

Fixed-point binary arithmetic is present in almost all computers. The floating-point format covers a wider range of numbers as compared to the fixed-point numbers. However, the floating-point arithmetic hardware is complex and costly. Hence, many old computers handled floating-point data by conversion to fixed-point data. Today's microprocessors have built-in floating-point units, thanks to low cost VLSI fabrication. Hence, nowadays the floating-point arithmetic is available free-of-cost.



4.3 Fixed-Point Numbers

In a fixed-point number, the binary point's position is fixed but not explicitly indicated in the computer. In majority of the computers, the binary point is assumed to be in the left of the most significant bit (msb). For example, the binary representation 1011 actually stands for 0.1011. Similarly 0100 denotes the number 0.0100. In some computers, the binary point is assumed to follow the least significant bit (lsb). In these computers, 1011 stands for 1011.0 and 0100 represents 0100.0. Hence, the fixed-point number 1010 may represent 0.1010 in one computer and 1010.0 in another. In the first case, it is a fraction whereas in the second, it is an integer.

4.3.1 Fixed-Point Number Representation Types

The fixed-point numbers may be signed or unsigned numbers. If a fixed-point number is given without sign (positive or negative), it is known as unsigned number. But inside a computer, a number always has sign. Usually, one bit indicates the sign. For a positive number, the sign bit is 0 and for a negative number, it is 1. There are three representation schemes for fixed-point numbers:

1. Signed magnitude form
2. 1's complement form
3. 2's complement form

Which representation is best? The two's complement scheme enables designing low-cost, high speed ALU. Hence computer designers choose the two's complement representation.

4.3.2 Signed Magnitude Form

The signed magnitude form is also known as true representation since the bit pattern is used without any change along with the sign bit. The following examples illustrate the signed magnitude form: (The position of the sign bit is indicated by the arrow)

Decimal value	Signed magnitude form
+3	011
-3	111
+1	001
-1	101
	↑
	Sign bit

Suppose we have 1 bit for sign and 2 bits for magnitude, the range of numbers that can be represented by this 3 bit system in a signed magnitude form, is from -4 to $+3$ as shown:

Binary pattern		Decimal value
Sign	Magnitude	
0	00	+ 0
0	01	+ 1
0	10	+ 2
0	11	+ 3
1	00	- 4
1	01	- 1
1	10	- 2
1	11	- 3

If n bits are used to represent the magnitude, the range of numbers that can be represented by the signed magnitude representation is (-2^n) to $(+2^n - 1)$. It is observed that the negative maximum is one more than positive maximum, due to use of the 00 combination of the magnitude bits, when the sign is negative. If the word length is 8 bits, the range that

can be represented is -128 to $+127$ since 1 bit is occupied by sign bit and remaining 7 bits are available for the magnitude. In signed magnitude form, the negative maximum number is 10000000 and its decimal value is -128 . The positive maximum number is 01111111 and its decimal value is $+127$. Though, the signed magnitude representation is a simple system, it is not suitable for arithmetic operations.

4.3.3 1's Complement Form

In this type of representation, the magnitude portion is shown in the complement form. The 1's complement of a number is obtained by complementing each bit: all 1's are made 0's and all 0's are made 1's. The following examples illustrate 1's complement formation:

Magnitude	1's complement
0111	1000
1000	0111
1001	0110

In 1's complement representation, a positive number is represented without any change, similar to signed magnitude form. But the negative number representation differs. The sign bit is 1 and the magnitude portion is put as 1's complement. The following examples illustrate this:

Decimal number	1's complement representation
+7	00111
-7	11000
	↑
	Sign bit

4.3.4 2's Complement Form

The 2's complement of a number is attained by adding 1 to the 1's complement of that number. The following examples illustrate 2's complement formation:

Magnitude	1's complement	2's complement
0111	1000	1001
1000	0111	1000
1001	0110	0111

In 2's complement number system, a positive number is represented in the original form, similar to the signed magnitude form. In the case of negative number, the sign bit is 1 and the magnitude is put as 2's complement. This is explained by the following examples:

Decimal number	2's complement representation
+ 7	00111
- 7	11001
	↑
	Sign bit

The 2's complement representation are very useful for the arithmetic operations as discussed in Chapter 5.

4.4 Floating-Point Numbers

In mathematics, real numbers are the numbers with fractions. For example, the decimal number 3.25 is a real number. It is same as $3\frac{1}{4}$. Inside a computer, the real numbers are represented in floating-point notation.

The floating-point number representation has three parts:

1. Mantissa
2. Base
3. Exponent (characteristic)

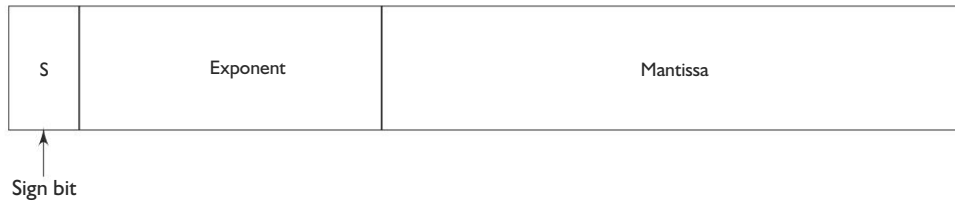
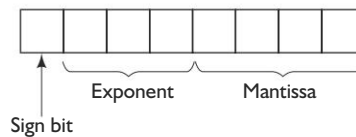
The following examples illustrate floating-point number system:

Number	Mantissa	Base	Exponent
3×10^6	3	10	6
110×2^8	110	2	8
6132.784	0.6132784	10	4
34.58	0.3458	10	2

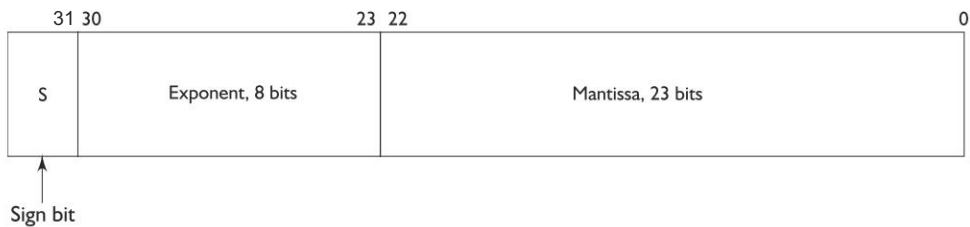
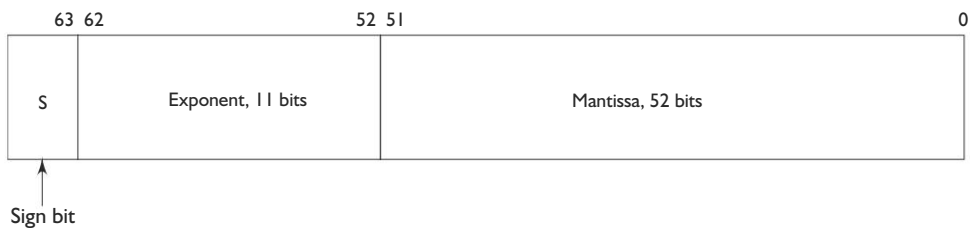
The mantissa and exponent are explicitly represented in the computer. But the base is implied for a given computer. Generally, the computers follow the base of 2. In general, a number f is represented as $f = m \times r^e$ where m is the mantissa, r is the base of the number system and e is the exponent (the power to which the base is raised). Figure 4.2 shows a general floating-point number format. A typical format for an 8-bit computer is shown in fig. 4.3. Consider the 8-bit floating-point number 00111101. The sign bit is 0, the exponent is 011 and the mantissa is 1101. The mantissa is actually .1101 since the radix point precedes the mantissa. The exponent is 011 (i.e. decimal 3). Since the sign bit is 0, the value is positive. The exponent of 011 indicates that the radix point in the mantissa should be moved three bit positions to the right. Hence the real value is 110.1.

In order to optimize the number of bits used in the floating-point representation, three standard concepts are followed:

1. Normalization of the number
2. Implicit leading bit for the mantissa
3. Biased exponent

**Fig. 4.2** Floating-point number format**Fig. 4.3** A Typical 8-bit floating-point format

These concepts are discussed in section 4.4.1 and 4.4.2. Initially, different computer manufacturers were using different formats (exponent, base etc.) for the floating-point representation. Presently the ANSI/IEEE standard is widely followed. It has two formats as shown in Figs. 4.4 and 4.5 for single and double precision.

**Fig. 4.4** Single precision floating-point number format-IEEE 754**Fig. 4.5** Double precision floating-point number format-IEEE 754

4.4.1 Normalization

A decimal floating-point number is in normalized form when the decimal point is placed to the right of the first non-zero mantissa digit. Similarly a binary floating-point number is in normalized form when the binary point is placed to the right of the first non-zero bit in the

mantissa. A number in unnormalized form can be converted into normalized form by shifting the radix point and adjusting the exponent. In other words, a floating-point number can be normalized by incrementing or decrementing the exponent so as to eliminate the leading zeros from the mantissa. If the binary point is shifted left by one bit, the exponent has to be incremented by 1. Similarly, if the binary point is shifted right by one bit, the exponent has to be decremented by 1.

Consider the number 0.011×2^{12} . It is not in normalized form. By shifting the binary point right by two bits, and decrementing the exponent by 2, we get 1.1×2^{10} .

This number is in normalized form.

Implicit leading bit

A binary floating-point number can be normalized by shifting the mantissa to eliminate the leading zeros and placing the binary point to the right of the first non-zero mantissa bit. As a convention, there is one bit to the left of the binary point. Since all normalized numbers have 1 as the most significant bit, we can save a bit position by treating it as implied or implicit leading bit by omitting it in storage. Hence to represent 1.1×2^{10} we store $.1 \times 2^{10}$ only. This is known as *hidden 1 principle* or 'implicit leading bit'. Due to this, a normalized number can effectively have 24-bit mantissa in IEEE 754 single precision format though we have 23 bits only for the mantissa. Similarly, it can have 53-bit mantissa in IEEE 754 double precision format though we have 52 bits only for the mantissa.

All zeros in mantissa represent the value 0. For any other number, hidden 1 form is used as shown in Fig.4.6(a) and Fig.4.6(b) for two numbers 1.0×2^{-2} and $1.0 \times 2^{+2}$ respectively.

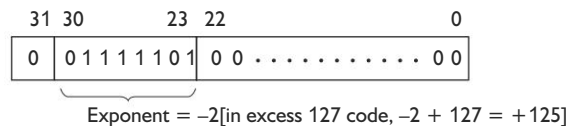


Fig. 4.6(a) Floating-point representation of 1.0×2^{-2}

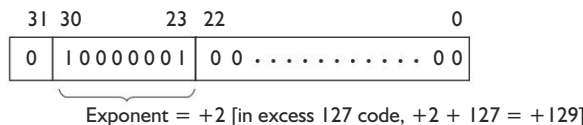


Fig. 4.6(b) Floating-point representation of $1.0 \times 2^{+2}$

4.4.2 Biased Exponent

We need to represent both positive and negative numbers in floating-point format. The exponent can be either a positive number or a negative number. How to represent the negative exponent? There are two standard options : true form or complement form. Another clever method is using a biased representation of the exponent. Consider the IEEE 754 single precision format. There are 8 bits for the exponent. Hence 256 combinations are possible. With 8 bits, the range covered by the exponent is -126 to $+127$ since two combinations are reserved for positive 0 and negative 0. In excess-code concept, a bias of 127 is added to the exponent. Actually the actual exponent is added to 127 and the result is used as the excess-127 code exponent. The advantage of using the excess-127 code is simplicity of expressing the negative exponents.

In excess-127 code, in the case of IEEE 754 single precision format, the negative exponent will be between 1 and 126 and the positive exponent will range from 128 to 255. Thus the exponent range of -126 to $+127$ gets converted into the $+1$ to $+254$ in excess 127 code format. Here are two special values: 0 and 255 in this concept. When both the exponent (in excess 127 code) and the mantissa are zero, this combination represents value 0. When the exponent (in excess 127 code) is 255 and the mantissa is zero, this combination represents ∞ . The ∞ is the result when any number is divided by 0.

In double precision format, 'excess 1023' code is used for expressing the exponent. The actual exponent is added with 1023 to get the contents of exponent field. The actual exponent is in the range -1022 to $+1023$ since we have 11 bits for the exponent field. This gets converted into $+1$ to $+2046$ in excess 1023 code. In double precision format, 0 and 2047 are reserved for special values for the exponent. As in excess-127 code, the combination of both the exponent and the mantissa being zero represents the value 0. When the exponent is 2047 and the mantissa is zero, this combination represents ∞ .

Example 4.1 Show the IEEE 754 binary representation of the number 5.25 in single precision floating-point format.

Step 1: Conversion of the decimal number into binary.

The integer part 5 is 101 in binary. The fraction part of 25 becomes 01 in binary. Hence equivalent the binary number is 101.01. It can be represented 0.10101×2^3 in scientific notation.

Step 2: Conversion of the binary number into Normalized number if already not in normalized form.

The binary number 0.10101×2^3 is in unnormalized form. By shifting the binary point right by 1 position, we get 1.0101×2^2 which is in normalized form.

Step 3: Single precision binary representation.

Sign bit is 0 since this is a positive number. The exponent is 2; adding 127, the excess 127 code exponent is 129. The binary equivalent is 10000001. The mantissa is 0101000...0. Hence the single precision format is shown in Fig.4.7

0 10000001 010100000000000000000000

Fig. 4.7 Single precision representation of value 5.25

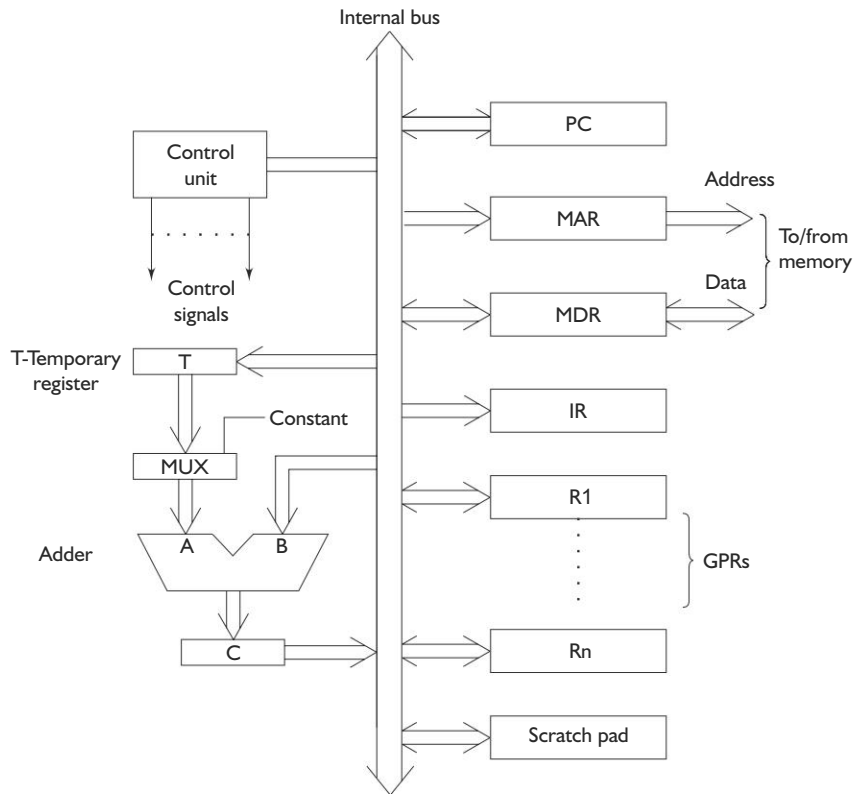
4.5 Fixed-Point Datapath

Binary fixed-point arithmetic operations are most important in the ALU. Four types of operations are needed: addition, subtraction, multiplication and division. There are two different strategies for implementing them:

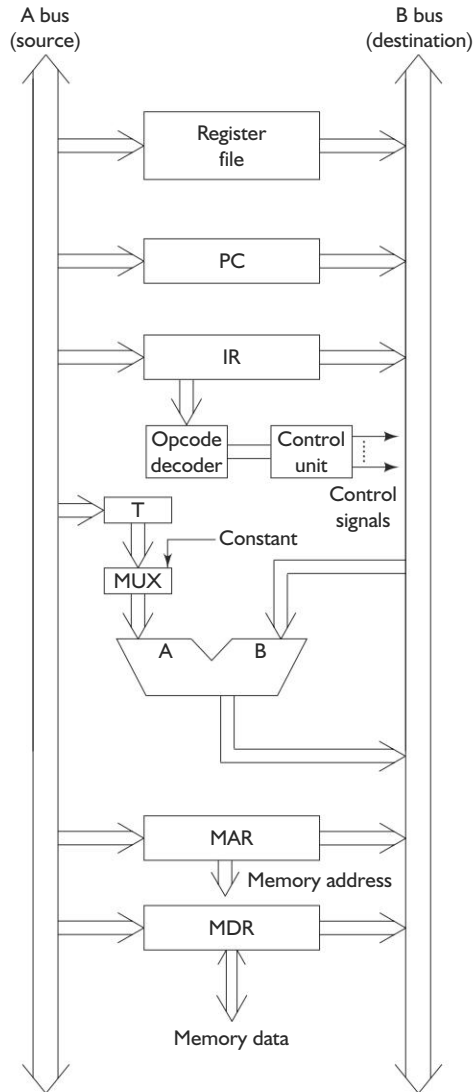
1. Designing ALU circuits for all the four operations. This approach leads to a costly but a fast ALU.
2. Designing ALU circuits only for addition and subtraction and the software performs multiplication and division. This approach leads to a slower but cheaper ALU.

The ALU consists of an adder and associated circuits such as counter, shifter, multiplexer etc. In a small CPU, the number of registers and other hardware components will be limited and a single internal bus interconnects the components of the datapath. Figure 4.8 shows the organization of a simple datapath. The speed of such a datapath is limited since different micro-operations involving the movement of data have to be done sequentially one after another through the internal bus. In a large CPU, where performance is the main concern, there is more than one bus for internal interconnections, and also there are a large number of registers and a variety of hardware circuits to perform several operations, simultaneously. Figure 4.9 shows a datapath with two internal busses. The A bus is the source bus for the ALU and the B bus is the destination bus for the ALU. Figure 4.10 shows the organization of a high performance datapath. It shows three internal busses. The A and B bus are input sources for the adder whereas the C bus carries the output of the adder.

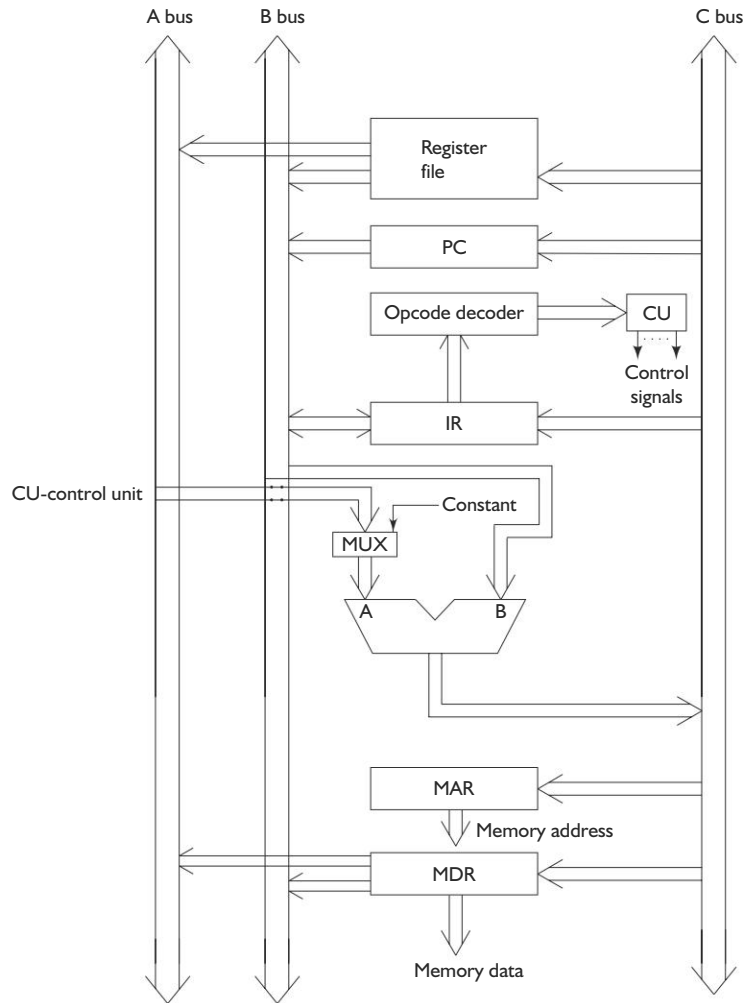
The datapath has a large number of control points which are activated by the control signals issued by the control unit. During the instruction cycle (i.e. fetching and executing an instruction), the control unit issues a sequence of control signals with different time delays depending on the current instruction. When a control signal activates a control point, the corresponding microoperation is executed by the datapath. Some of the common microoperations are as follows:

**Fig. 4.8** Single bus based datapath

1. Shift a register contents
2. Increment a counter
3. Decrement a counter
4. Select one signal out of many signals (multiplexing)
5. Add the inputs of the adder
6. Copy the contents of a register into another (transfer)
7. Complement a register contents
8. Clear a register contents (reset)
9. Set a flip-flop
10. Reset a flip-flop
11. Enter a constant into a register (preset)
12. Read from main memory

**Fig. 4.9** Two bus processor datapath

13. Write into main memory
14. Read from an input port
15. Write into an output port

**Fig. 4.10** Three bus based datapath

The purpose of the micro-operations can be fully understood if the algorithms used for the arithmetic operations are known. In a given CPU, it is possible to implement an arithmetic operation either fully by the datapath or jointly by the datapath and the control unit. For example, consider a multiplication operation. The ALU can have a multiplier hardware circuit. Alternatively, the ALU can have only addition and subtraction hardware and the multiplication can be achieved by a sequence of additions and left shifts. The control unit generates the required control signals in the required time sequence for the datapath. In the following sections, the circuits for the arithmetic operations are discussed first and then the datapath design for some simple instructions is discussed.

Example 4.2 Design an incrementer circuit to increment the contents of a three bit register.

There are two ways of designing the incrementer:

1. Using a binary up counter. A counter consists usually of J-K flip-flops and for each clock signal, the up counter increments by 1. The counter may be asynchronous counter or synchronous counter.

2. Using a combinational circuit. Fig. 4.11 shows the block diagram for the incrementer using combinational circuits. Table 4.1 shows the truth table for the incrementer. The truth table shows, in a tabular form, the relationship between the output and the input. The table covers all possible input combinations and corresponding output. Each row in the truth table represents one of the input combinations. Figure 4.12 shows the logic diagram which satisfies the truth table. Another way of looking at the incrementer is addition of 001 to the given number. Hence, we can make use of an adder with 001 as one input. Figs 4.13 and 4.14 show configuration of half-adder and full adder as incrementer.

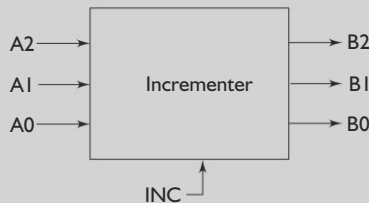


Fig. 4.11 Incrementer block diagram

TABLE 4.1 Function table for incrementer

Input combination A2 A1 A0	Output combination B2 B1 B0
0 0 0	0 0 1
0 0 1	0 1 0
0 1 0	0 1 1
0 1 1	1 0 0
1 0 0	1 0 1
1 0 1	1 1 0
1 1 0	1 1 1
1 1 1	0 0 0

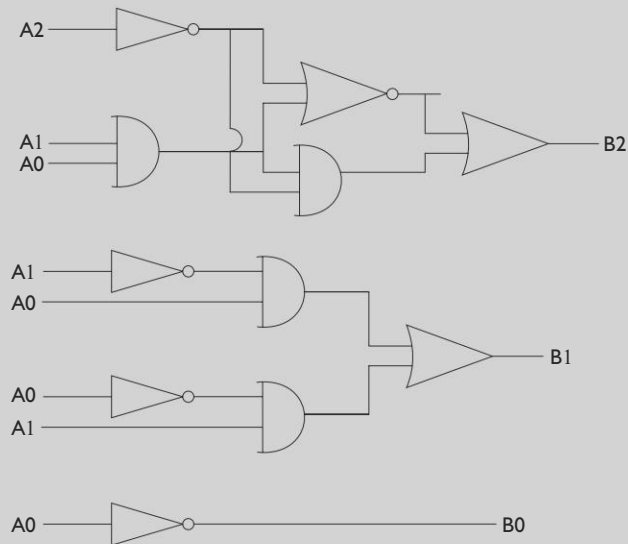


Fig. 4.12 Incrementer logic diagram

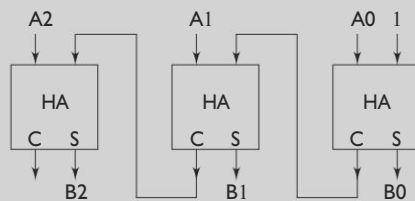


Fig. 4.13 Incrementer using half adders

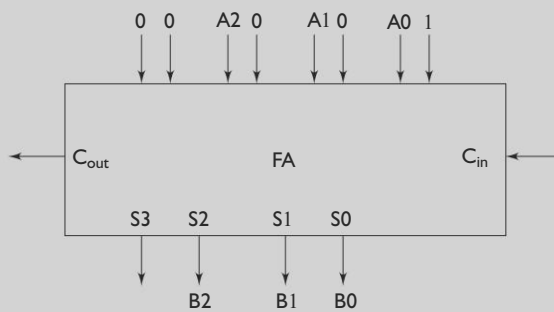


Fig. 4.14 Incrementer using full adder

4.6 Design of Arithmetic Unit

An arithmetic unit performs several arithmetic micro-operations such as addition, subtraction, complementation, incrementation, decrementation etc. Figure 4.15 illustrate a block diagram of an arithmetic unit. Using a single parallel adder, all these micro-operations can

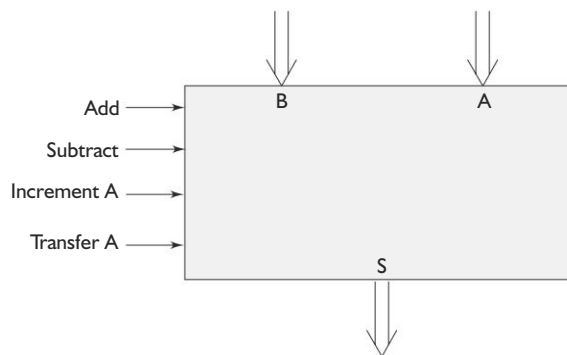


Fig. 4.15 Block diagram of an arithmetic unit

be performed by applying appropriate input to the adder as shown in Fig. 4.16. As noticed from this logic diagram, the four numbers of 4 to 1 multiplexors do the routing of



Fig. 4.16 Logic diagram of an arithmetic unit

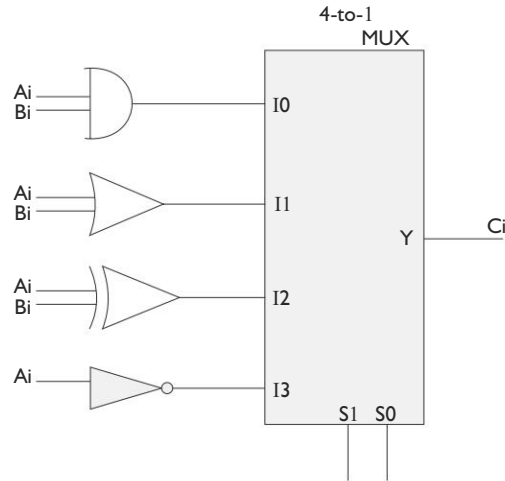
appropriate input to Y input of the adder. Table 4.2 shows eight combinations of S_1 , S_0 and C_{in} and corresponding Y input of adder and the adder output. The X input of the adder is the first operand, A. Thus, the required arithmetic micro-operation can be performed by applying the corresponding signal inputs to S_1 , S_0 and C_{in} . Though there are eight input patterns, this arithmetic circuit performs only seven distinct micro-operations. Cases 5 and 8 perform the same micro-operation: transfer A to S. Any one of these two combinations can be chosen in the design. The seventh case to has be analyzed carefully. The input combination is $S_1, S_0, C_{in} = 110$. The I_3 input of MUX is selected thus, supplying all 1's to Y input of the adder. All 1's such as 1, 11, 111, 1111 etc. is nothing but 2's complement of 1, 01, 001, 0001, respectively. Thus, the 7th case adds 2's complement of 1 to X. This is equivalent of subtracting 1 from X. In other words, the micro-operation performed is 'decrement A'.

TABLE 4.2 Arithmetic unit function table

S. No.	S_1	S_0	C_{in}	Y input to MUX	MUX output	Operation
1	0	0	0	B	$A + B$	Add
2	0	0	1	B	$A + B + 1$	Add with carry
3	0	1	0	$\bar{B}$	$A + \bar{B}$	Subtract with borrow
4	0	1	1	$\bar{B}$	$A + \bar{B} + 1$	Subtract
5	1	0	0	0	A	Transfer A
6	1	0	1	0	$A + 1$	Increment A
7	1	1	0	1	$A - 1$	Decrement A
8	1	1	1	1	A	Transfer A

4.7 Design of Logic Unit

Using a 4-to-1 MUX to each bit position, a logical unit can be designed by giving appropriate input to the four inputs of the MUX. Figure 4.17 shows the design of one bit position which performs four logic operations: AND, OR, XOR and complement. Table 4.3 shows the input combinations and corresponding output and logical operation performed on the operands A and B.

**Fig. 4.17** One bit of logic unit**TABLE 4.3** Logic unit function table

S. No.	S1	S0	C	Logic operation
1	0	0	$A \wedge B$	AND
2	0	1	$A \vee B$	OR
3	1	0	$A \oplus B$	Exclusive OR
4	1	1	$\overline{A}$	Complement A

4.8 Design of Shifter

Shift operation is needed in ALU for several reasons: data processing, multiplication, division, etc. There are two approaches to perform shift operation on the contents of a register:

1. A separate combinational shift register can act as a centralized resource. The contents of the register to be shifted are transferred to input of the shift register. The shift register outputs are given back to the original register. Figure 4.18 shows how the contents of R1 are shifted by making use of the shift register K. The micro-operation $K := R1$ transfers the contents of R1 to the shift register K. Another micro-operation, $R1 := K$ is performed after the propagation delay of the shift register. Now the shift register output are transferred to R1.

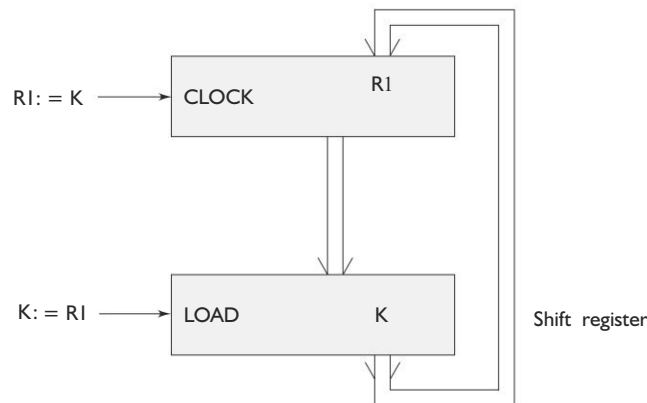
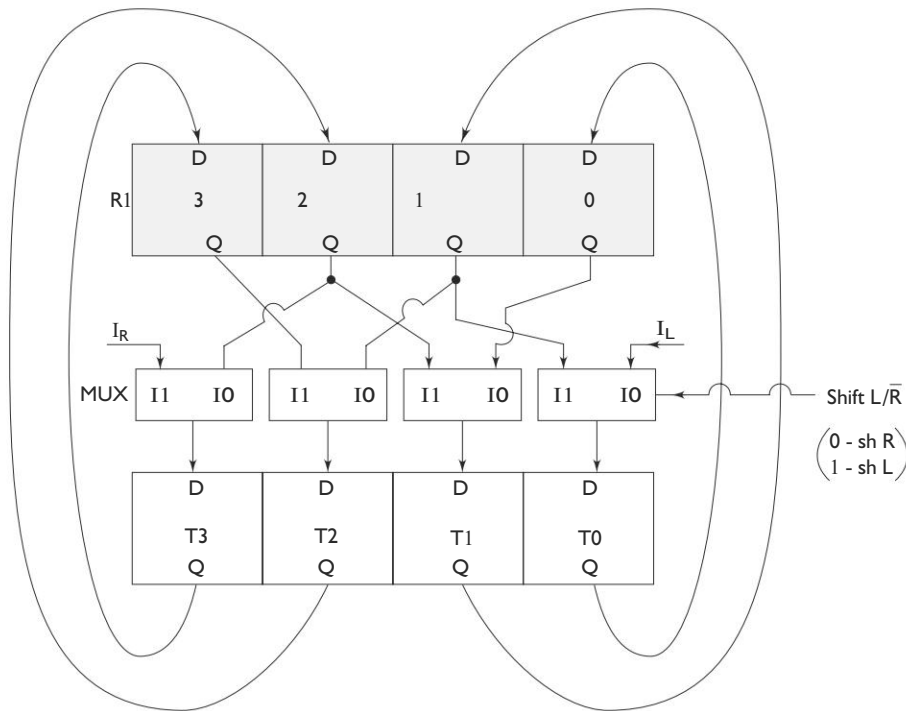


Fig. 4.18 Use of separate shifter

2. A dual bank register should be designed as a shift register so that the shift operation is performed in the same register. This involves additional hardware circuits requirement in register R1. Figure 4.19 shows how a register R1 is modified into a shift register. The design has two set of flip-flops for each bit of R1. The flip-flops T0, T1, T2 and T3 act as a temporary set for the corresponding bits R0, R1, R2 and R3. A 2×2 -to-1 MUX supplies to the D input of each T flip-flop. One of the inputs to the MUX correspond to the left shift requirement and the other input to the right shift. The select input to the MUX indicate the shift direction: left or right. Table 4.4 shows the function table for the shift register. In effect, the R1 is now a dual bank register.



Note: The four 2-to-1 MUX have common select input controlled by shift $L/\bar{R}$ -signal.

Fig. 4.19 Shift register circuit

TABLE 4.4 Shift register truth table

S. No.	Shift $L/\bar{R}$	T3	T2	T1	T0
1	1	$R\ 1(2)$	$R\ 1(1)$	$R\ 1(0)$	I_L
2	0	I_R	$R\ 1(3)$	$R\ 1(2)$	$R\ 1(1)$

4.9 ALU Design

An ALU can be either a combinational or sequential ALU. A combinational ALU is a combinational circuit that can perform arithmetic, logical and shift operations. The combinational ALU can be designed by combining the designs of arithmetic unit, logic unit and shifter seen in Sections 4.6–4.8. Figure 4.20(a) shows a block diagram of a combinational ALU. A 4-bit combinational ALU on a single IC is available as 74181. By cascading multiple ICs of 74181, ALUs of longer word length can be built. It is possible to include

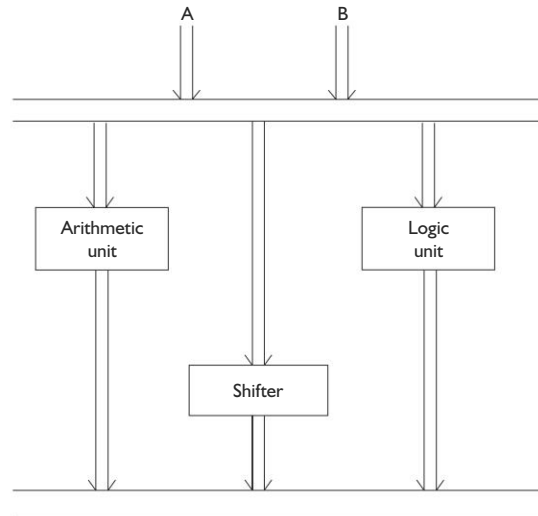


Fig. 4.20(a) Block diagram—bus oriented combinational ALU

combinational multipliers and dividers but they need several logic levels. Hence, they are costly and slow. In comparison the sequential ALUs are cheaper. They implement multiplication and division using algorithms discussed in Chapter 5. Figure 4.20(b) shows

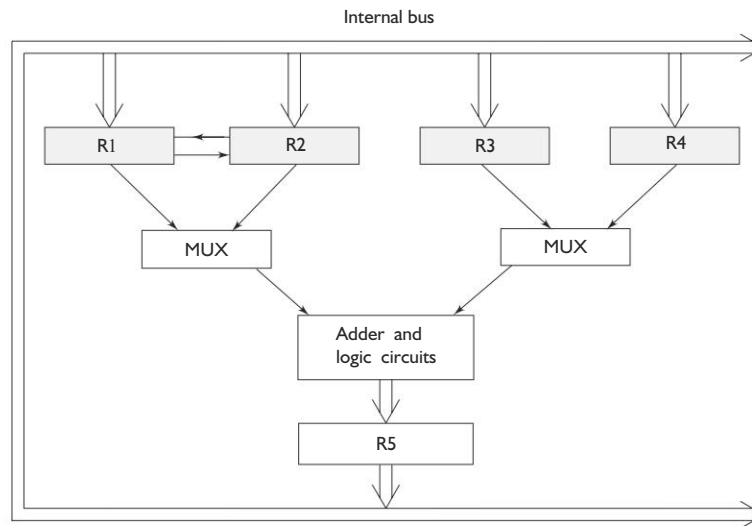


Fig. 4.20(b) Sequential ALU

R2—Multiplier/quotient register

R3—Multiplicand/divisor register

R1, R2—Product/dividend

a sequential ALU. The complete 4-bit sequential ALU is provided by ICs AMD 2901 and AMD 2903. These are micro-programmable bit slice processors. Figure 4.21 shows a block diagram of AMD 2903.

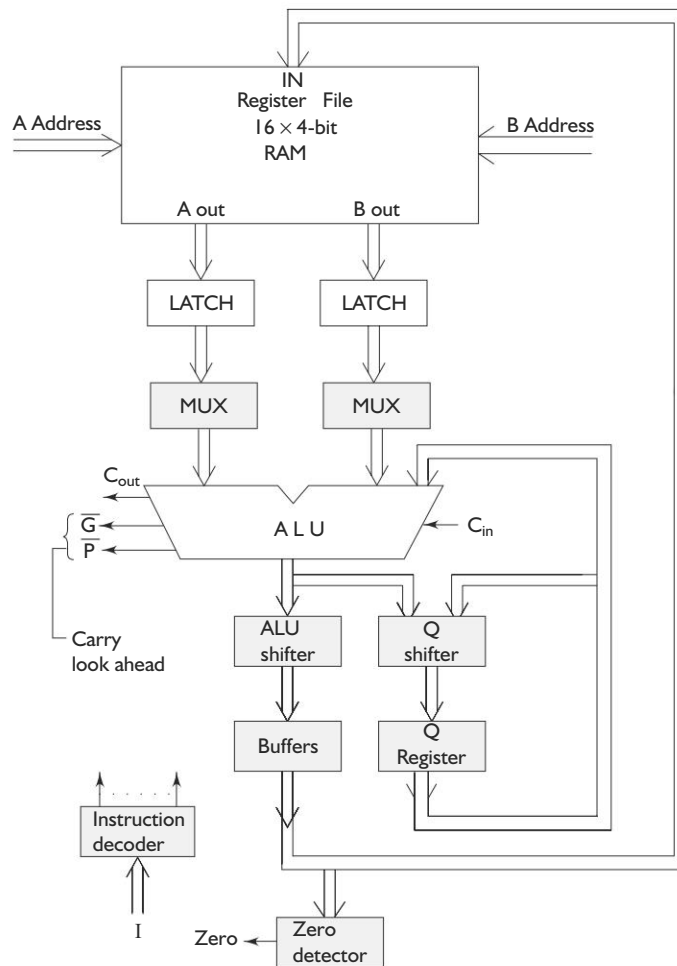


Fig. 4.21 Block diagram of AMD 2903



4.10 Typical Minicomputer Datapath

Figure 4.22 shows the datapath of PDP-11 minicomputer. The A leg and B leg are input to the ALU in which most of the data processing takes place.

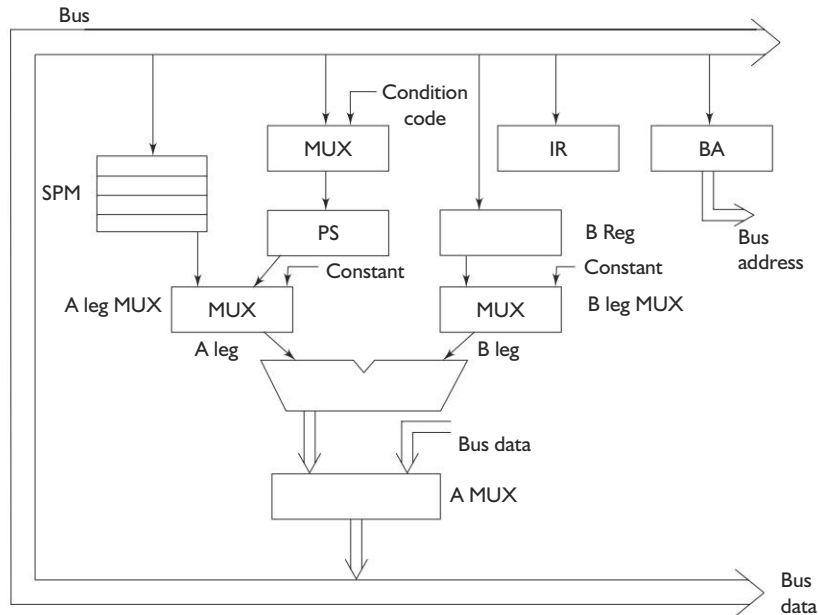


Fig. 4.22 Datapath of PDP-11 minicomputer

The A leg MUX has three input sources:

1. Scratch Pad Memory (SPM) which is a register array. It contains both program addressable registers and non-addressable internal registers.
2. Program Status (PS) register
3. Constant

The B leg MUX has two sources:

1. B register
2. Constant

The B leg MUX can be used for the following:

1. Routing one operand
2. Performing sign extension and other functions on the operand on its way to the ALU

The AMUX selects either the ALU output or the bus data. Its output can be sent to most of the processor registers.

The IR is the instruction register that holds the current instruction. The BA contains the current address placed by the processor on the bus.

4.11 Typical Mainframe Datapath

Figure 4.23 shows the datapath of a IBM System/360 compatible CPU. The IBM System/360 had many models and some of the models have been emulated by some non-IBM companies. This datapath shows both the fixed-point ALU and logic units. The floating-point datapath is not included in this diagram. The R5 is also used as MDR and R4 is used as MAR. The counters 1 and 2 are used for different counting purposes. The AMUX can shift the adder output whenever required. This datapath presents a sequential ALU for performing multiplication and division operations using algorithms discussed in Chapter 4. The use of different registers in multiplication and division are as follows:

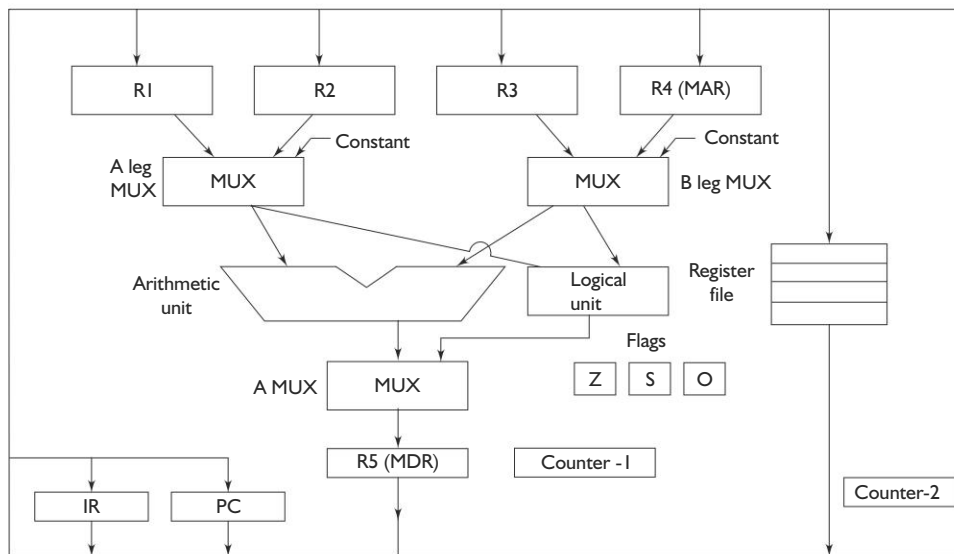


Fig. 4.23 Datapath of a typical mainframe CPU

Multiplication

Multiplier: R2

Multiplicand: R3

Product: R1 and R2

Division:

Dividend: R1 and R2

Divisor: R3

Quotient: R2

Remainder: R4

4.12 Main Memory Interface

The interface between the main memory and the processor is shown in Fig. 4.24. The MR/W flag is used by the processor to indicate the exact operation it is performing. If it is 0, the processor does memory read operation. If it is 1, the CPU does memory write operation. The SMMA flag indicates whether a memory operation is already in progress. If it is 0, no memory operation is in progress. If it is 1, a memory operation is in progress. When the processor changes the state of SMMA from 0 to 1, the memory starts the operation. On completion of the operation, the memory control logic resets the flag. Obviously, this will be done after the memory access time is over. The MFC signal from memory can be used to reset this flag.

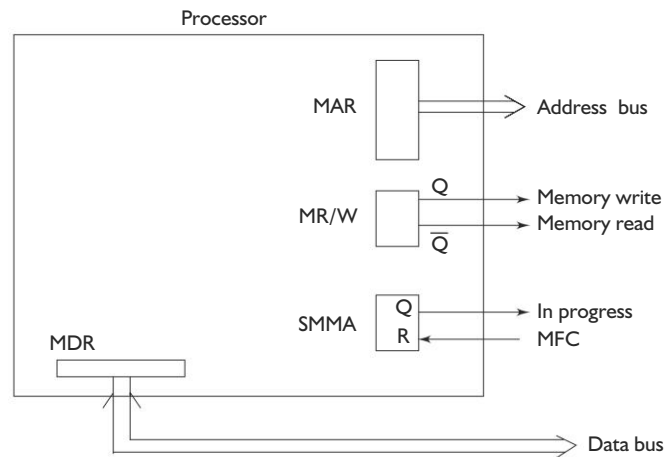


Fig. 4.24 Main memory interface

4.12.1 Memory Read Sequence

The processor does following operations for performing a memory read:

1. Puts the memory address in MAR.
2. Resets MR/W flip-flop. Its $\overline{Q}$ output goes as MEMORY READ control signal.
3. Sets SMMA flag.
4. Keeps checking whether SMMA flag has become 0. Once the SMMA becomes 0, the processor loads the data from memory into MDR.

4.12.2 Memory Write Sequence

The processor does following operations for performing a memory write:

1. Puts the memory address in MAR.
2. Puts data in MDR.
3. Sets MR/W flip-flop. Its Q output goes as MEMORY WRITE control signal.
4. Sets SMMA flag.
5. Keeps checking whether SMMA flag has become 0. Once the SMMA becomes 0, the processor proceeds with other operations.



4.13 Local Storage/Register File

The local storage consists of a set of processor registers (with a common read/write control) also known as registers file. There are two types of registers:

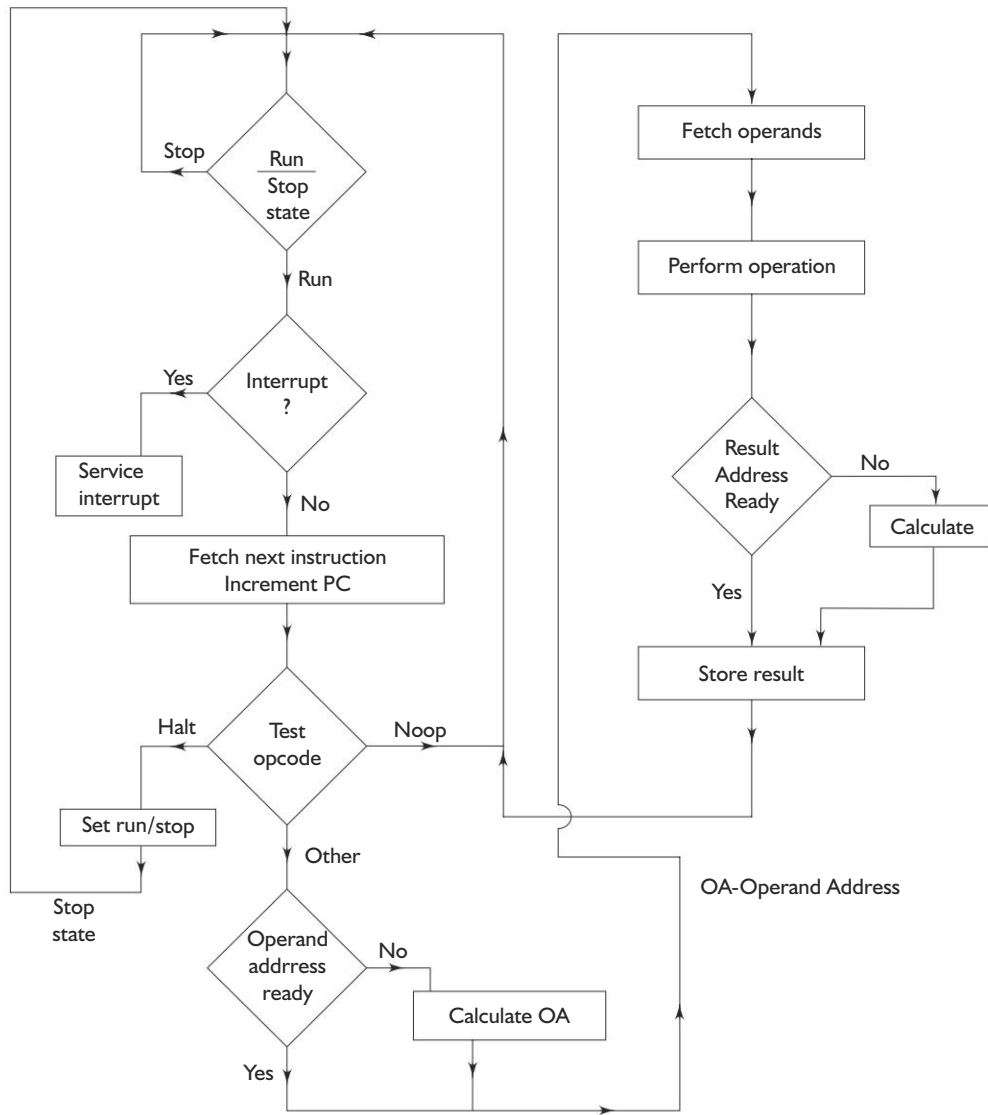
1. Program addressable registers for storing the operands or results. These are known as general purpose registers. They can be also used as special registers such as index register, base register, stack pointer etc.
2. Scratch pad registers used as temporary registers by the control unit for storing the intermediate results or constants required during an instruction execution. These registers are invisible to the program.

The access time of the local storage is very small as compared to access time of the main memory. Physically, the local storage is inside the processor. It should be noted that the local storage is different from cache memory which is a buffer between the processor and main memory. The cache memory is invisible to programs. Another important observation here is that the program (however small) cannot be stored in the local storage since the processor does not fetch instruction from the local storage.



4.14 Datapath for Simple Instructions

The datapath for instruction fetch is common for all the instructions. Figure 4.25 shows the fetch sequence and Fig. 4.26 shows the corresponding datapath that consists of PC, MAR, MDR and IR. The control points and the required control signal sequence are also indicated in the same. At the end of the fetch, the contents of PC is incremented by the byte length of the currently fetched instruction so that the PC points to the next instruction address. This is an advance preparation for the next instruction fetch.

**Fig. 4.25** Instruction fetch sequence

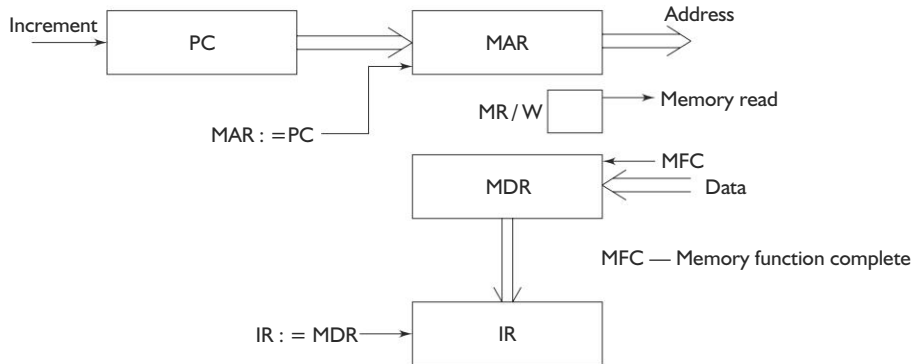


Fig. 4.26 Datapath for instruction fetch

4.14.1 HALT Instruction

The action done by the HALT instruction is stopping the instruction cycle. This is described as $RUN/HLT := 0$. In terms of data path, it involves resetting the RUN/HLT flip-flop. Figure 4.27 shows the datapath. The control signal $CLEAR\ RH$ activates the control point which is the clock input of the D flip-flop. The D input is permanently pulled down (0). Hence, on leading edge of the $CLEAR\ RH$ signal, the flip-flop is reset. The Q output of the flip-flop is sensed by the control unit before commencing an instruction fetch. If it is 0, instruction fetch is not done. The control unit simply keeps on waiting for the RUN/HLT flip-flop to become set again. This is done by any of the following two actions:

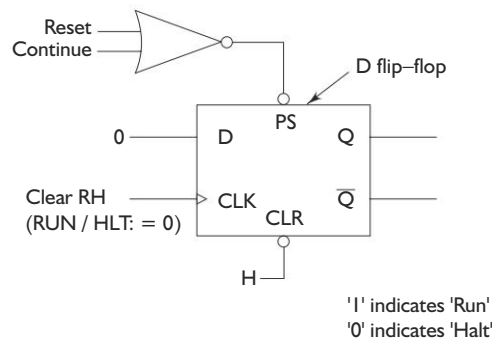


Fig. 4.27 Datapath for HALT instruction

1. RESET signal becomes active (HIGH)
2. CONTINUE signal becomes active (HIGH)

The NOR gate generates a LOW output signal that is given to the preset input of the flip-flop.

The RESET signal is generated during power on time and during manual reset time due to operator pressing RESET button/switch on the front-panel. The CONTINUE signal is a special signal which may not be present in a simple computer. It is present in special systems such as multi-processor systems.

4.14.2 NOOP Instruction

No action is done by the NOOP instruction. Hence, no datapath exists for this instruction. As soon as this instruction is decoded, the execute phase is completed.

4.14.3 JUMP Instruction

The JUMP instruction causes branch to the instruction address, specified in the JUMP instruction. This is described as $PC := BA$ where BA is the branch address given by the instruction. This address is usually calculated according to the addressing mode in the instruction. Assuming that the address calculation is completed and BA is available in ALU register R5, the micro-operation desired is $PC := R5$. Figure 4.28 shows the datapath for JUMP instruction.

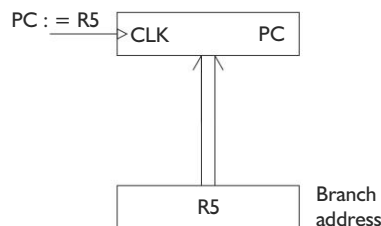


Fig. 4.28 Datapath for JUMP instruction

4.14.4 LOAD Instruction

The action performed by the LOAD instruction is to copy a memory location content into a register, specified in the instruction. This is described as $\langle RA \rangle := \langle MA \rangle$ where MA is the memory address and RA is the register address. The sequence of operations required are as follows:

4.14.5 STORE Instruction

The action done by the STORE instruction is reverse of the action by LOAD instruction. A register content is copied into a main memory location. This is described as $\langle \text{MA} \rangle := \langle \text{RA} \rangle$. The required sequence of operations is as follows:

1. Read from the local storage, the contents of register given by RA.
2. Write in main memory location MA.

The datapath organization is shown in Fig. 4.30. At the end of instruction fetch, the instruction is available in IR. Let us assume that bits 0–7 gives the opcode, bits 8–11 gives RA and bits 12–23 gives MA.

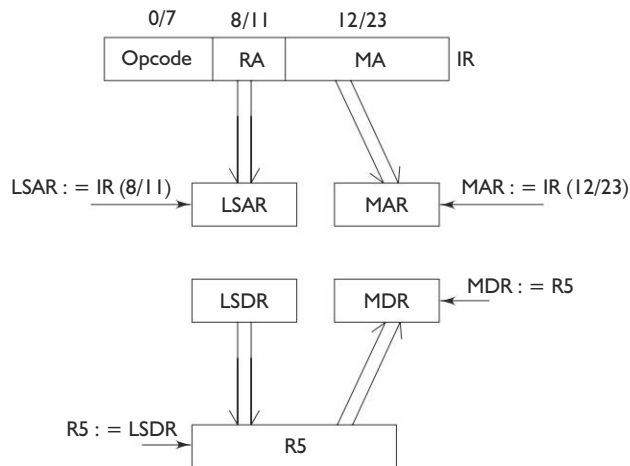


Fig. 4.30 Datapath for STORE instruction

The register transfer statements are given below along with comments:

- | | |
|----------------------|--|
| 1. LSAR := IR (8/11) | Transfers RA into local storage address register |
| 2. LSR/W := 0 | Resets the local storage read/write flag; indicates read |
| 3. SLISA := 1 | Sets start local storage access flag |
| 4. R5 := LSDR | Transfers local storage data register contents into R5 |
| 5. MDR := R5 | Transfers R5 contents into main memory data register |
| 6. MAR := IR (12/23) | Enters MA in main memory address register |
| 7. MR/W := 1 | Sets the memory read/write flag; indicates write |
| 8. SMMA := 1 | Sets start main memory access flag |

The control unit should ensure that the time delay between operations 3 and 4 is greater than local storage access time. The eight register transfer statements can be directly used as microoperations.

4.15 Floating-Point Unit Datapath

Figure 4.31 shows typical organization of floating-point unit generally found in medium range older computers. There are two sections:

1. Exponent unit
2. Mantissa unit

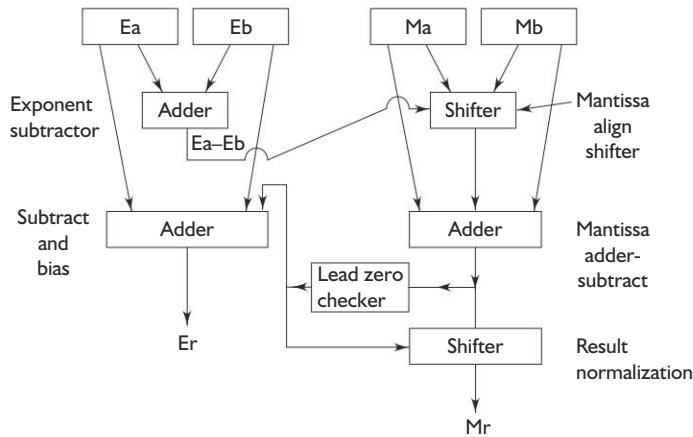


Fig. 4.31 Generic floating point unit

As discussed in Chapter 5, the following steps are done for floating-point addition or subtraction:

1. Exponent comparison
2. Mantissa alignment for one of the operands
3. Mantissa addition or subtraction

The mantissa unit is not necessarily an exclusive hardware. A floating-point unit which supports all four basic operations (add, sub, multiply and divide) can also serve as the mantissa unit. The exponent unit has to perform three different operations:

1. Addition of exponents
2. Subtraction of exponents
3. Comparison of exponents

The exponent adder is used to determine $E_a - E_b$. The larger exponent is identified based on the sign of this subtraction. The magnitude of difference (E) gives the number of shifts required to the mantissa of the number having larger exponent.

Figure 4.32 gives the organization of a high performance floating-point unit used in IBM System/370 model 195. It is an independent floating-point unit, without any dependence on the fixed-point ALU, for mantissa operations. This provides both operand buffering and instruction buffering. The FLOS is the operator stack. The FLR is the set of floating-point registers. The FLB is the set of operand buffers. There are three execution units. The reservation stations in each execution unit allows concurrent operations by the execution unit. The extended execution unit handles extended-precision floating-point operands.

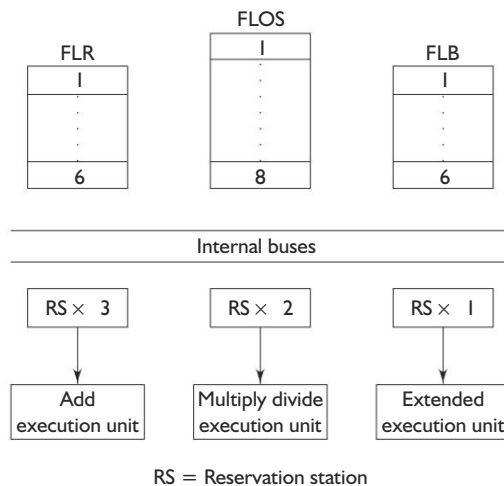


Fig. 4.32 Floating-point unit in IBM system/370 model 195

4.15.1 Floating-Point Processors as Coprocessors

An old trend in the microprocessor industry is making the floating-point unit as a special purpose processor which works with an integer processor as a coprocessor. Table 4.5 lists some Intel integer processors and their floating-point companion processors. For instance, INTEL 8087 is a coprocessor designed to work with INTEL 8088 (and INTEL 8086). It has the ability to execute a set of 68 floating-point arithmetic instructions. It is external to the integer processor but closely linked to it through the local bus of the integer processor. Advanced microprocessors such as 80486 DX and Pentium series have integrated coprocessors in the microprocessor chips itself. More later microprocessors such

as INTEL PENTIUM has several improvements in floating-point coprocessors such as pipelined FPU, hardware multiplier and divider etc. These features enhance the performance of the CPU.

TABLE 4.5 Intel integer processors and coprocessors

S. No.	Integer processor	Coprocessor
1	8086/8088	8087
2	80286	80287
3	80386	80387
4	80486 SX	80487 SX

4.15.2 Coprocessor Organization And Synchronization

Let us consider the original IBM PC (Personal Computer) which had the INTEL 8087 as an optional unit. In PCs, which had 8087, the floating-point instructions are executed by 8087. If 8087 is not installed in a PC, the fixed-point software that simulates the floating-point operations get control. Figure 4.33 shows the simplified block diagram of 8087. A register stack of eight 80-bit registers holds the operands. The 80-bit floating-point data format provides 18 decimal digit accuracy. The registers can be used in two ways:

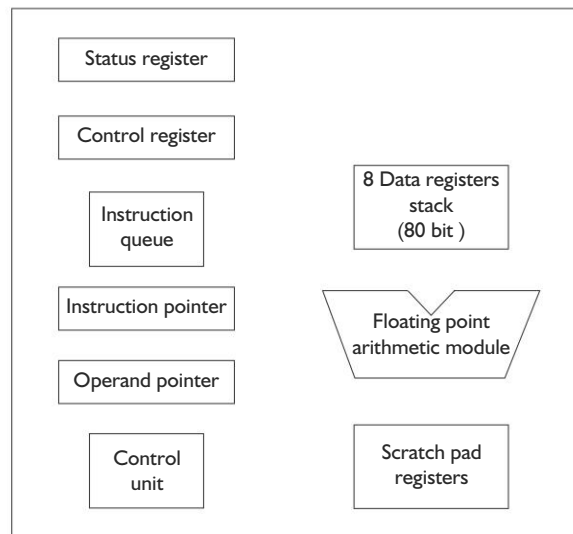


Fig. 4.33 8087 Block diagram

1. As a standard register set.
2. As a stack, for passing parameters to a floating-point subroutine. Stack mode addressing is also used.

Figure 4.34 shows the interface between 8088 and its floating-point coprocessor 8087. The program has instructions of both 8088 and 8087. Instructions are fetched and decoded by 8088. The 8087 also *steals* the instruction from the bus when 8088 is doing instruction fetch. It ignores the instruction if it belongs to 8088. The 8087 instruction has ESC code with 11011 pattern. On recognizing this, the 8087 executes the instruction whereas the 8088 keeps quiet. In some cases, the 8088 helps 8087 in fetching the first byte of the operand and the remaining bytes are fetched by 8087. To synchronize 8088 and 8087, a WAIT instruction is placed after a floating-point instruction. When 8087 is *busy* executing a floating-point instruction, the 8088 executes the WAIT instruction and keeps on sensing its TEST input to which the BUSY output of 8087 is connected. The BUSY signal indicates whether 8087 has completed the previous instruction or not. Once 8087 completes the instruction, the BUSY and the TEST signals become LOW. On sensing this, the 8088 goes to next instruction after the WAIT instruction.

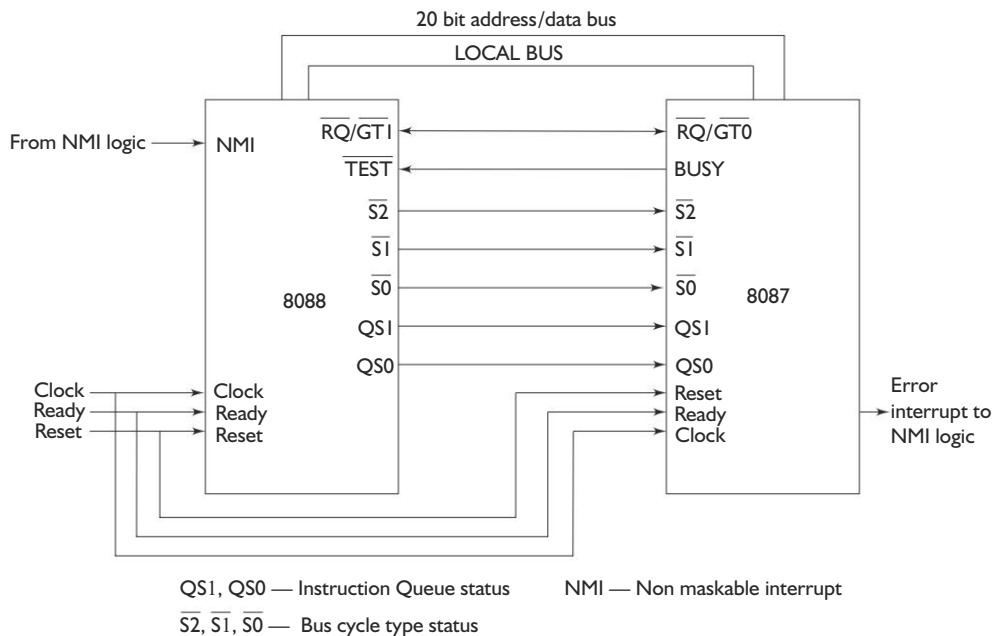


Fig. 4.34 Interface between 8088 and 8087 in the PC



4.16 Advanced Processors and Datapaths

Several high performance design approaches are followed in designing an advanced processor. Some of these are pipelining, superscalar architecture, out of order execution, branch prediction, register renaming etc. These topics are discussed later in Chapters 12 to 16.

SUMMARY

Fixed-point arithmetic is also known as integer arithmetic. In most of the computers, complementary arithmetic is followed for integer arithmetic. Real numbers can contain both whole and fractional parts. Inside the computer, the real numbers are generally represented in floating-point notation. The position of radix point is not fixed. It can be moved anywhere within a bit string. Hence this notation offers flexibility of manipulation.

The datapath includes following hardware components:

1. ALU.
2. Registers for temporary storage.
3. Various digital circuits for executing different micro-operations. These include hardware components such as gates, latches, flip-flops, multiplexers, decoders, counters, complementer, delay logic etc.
4. Internal path for movement of data between ALU and registers.
5. Driver circuits for transmitting signals to external units (control unit, memory, I/O).
6. Receiver circuits for incoming signals from external units.

The datapath has a large number of control points which are activated by the control signals issued by the control unit. During the instruction cycle (fetching and executing an instruction), the control unit issues a sequence of control signals with different time delays depending on the current instruction. When a control signal activates a control point, the corresponding micro-operation is executed by the datapath.

An ALU can be either a combinational or a sequential ALU. A combinational ALU is a combinational circuit that can perform arithmetic, logical and shift operations. The combinational ALU can be designed by combining the designs of arithmetic unit, logic unit and shifter. It is possible to include combinational multipliers and dividers also but they need several logic levels. Hence, they are costly and slow. On the other hand, sequential ALUs are cheaper. They implement multiplication and division using algorithms.

The floating-point unit ALU is totally independent or its mantissa unit uses fixed-point unit ALU. Another trend in the microprocessor industry is using the floating-point unit as a special purpose processor that works with an integer processor as a coprocessor. Advanced

microprocessors such as 80486 DX and Pentium series have integrated coprocessors in the microprocessor chips itself.

REVIEW QUESTIONS

1. Incrementing the program counter is done during instruction fetch so as to point to the next instruction address. This operation becomes dummy (unnecessary) in case the instruction fetched is a JUMP (branch) instruction. To eliminate this, we can suggest another approach. Instead of incrementing the program counter during the fetch-phase (subcycle), it can be done during the execution-phase along with the last microoperation, if only applicable. No computer seems to follow this approach. Identify the flaws in our suggestions. (*Hint: interrupts, cycle time*)
2. What is the use of the microoperation 'subtract with borrow' when we have 'subtract' microoperation?
3. A register contains an 8 bit binary data. It is required to complement the three lsbs only. Suggest the datapath hardware (out of the following list) where it can be carried out and identify the sequence of the microoperations involved:
 - (a) Arithmetic unit
 - (b) Logical unit
 - (c) Shifter
 - (d) Register file
 - (e) Internal bus
4. The time spent in instruction fetch for JUMP (unconditional branch) instruction can be saved if we redesign the instruction set as follows:
 - There should be no separate JUMP (or BUN) instruction. Instead the branch operation can be combined with any other instruction as follows:
ADDJUMP: Add and then jump (branch); LDAJUMP: Load accumulator and then branch.
 - We have to meet following additional requirements to implement this 'novel' idea:
 - (a) Instruction length has to be increased by 1 bit. This bit in the opcode can be used to differentiate normal (ADD) instruction from the new (ADDJUMP) instruction.
 - (b) Jump address can be indicated in one of the operand address field where as the other operand address field can be used for storing the result as usual.

Is this idea practically feasible? Will it require any change in the datapath? If, on an average, 20% of the instructions are unconditional branch instructions, which instruction set produces an efficient CPU—normal or revised?

5. Scratch pad registers are not addressable by instructions. The following can address them (strike out the wrong answers):
 - (a) Compiler
 - (b) Operating system
 - (c) Control unit
 - (d) Input unit
6. Consider a computer that follows 2's complement arithmetic. The user presents the data in the decimal form. The data is converted into 2's complement by whom?
 - (a) Operating system
 - (b) Compiler
 - (c) User program (application)
 - (d) System utility

Is it necessary to have an instruction (in the instruction set) for the decimal to binary conversion?
7. A computer has floating-point arithmetic instructions. But the CPU hardware (datapath) does not have the floating-point ALU. How should the system handle floating-point arithmetic?
8. In some computers, the floating-point ALU is self-contained and has no dependence on integer ALU. What are the advantages and disadvantages of such computers?

EXERCISES

1. Calculate the time taken for instruction fetch in a CPU which has the following propagation delays:
 - (a) Any register transfer operation: 10 ns
 - (b) main memory access time: 80 ns

Assuming zero delay from the completion of one microoperation to the beginning of the next, show the break-up of your calculations.
2. Which of the following instructions takes maximum time for instruction cycle?
 - (a) LDA
 - (b) STA
 - (c) NOOP
 - (d) HALT
 - (e) BUN (branch unconditionally)

Justify your answer with quantitative estimates.
3. Which of the instructions in Question 2 takes minimum time for instruction cycle? Justify your answer with quantitative estimates.
4. A memory location contains 1100001011110010. Find the decimal value of this binary word if it represents the following:
 - (a) Unsigned integer
 - (b) 1's complement integer
 - (c) 2's complement integer
 - (d) Signed magnitude integer

5. Convert following IEEE single-precision floating-point numbers to their decimal values:

(a) 00111111111100000000000000000000

(b) 01100110100110000000000000000000

6. Convert the following decimal numbers to IEEE single-precision floating-point format:

(a) -58

(b) 3.1823

(c) 0.6

(d) -38.24

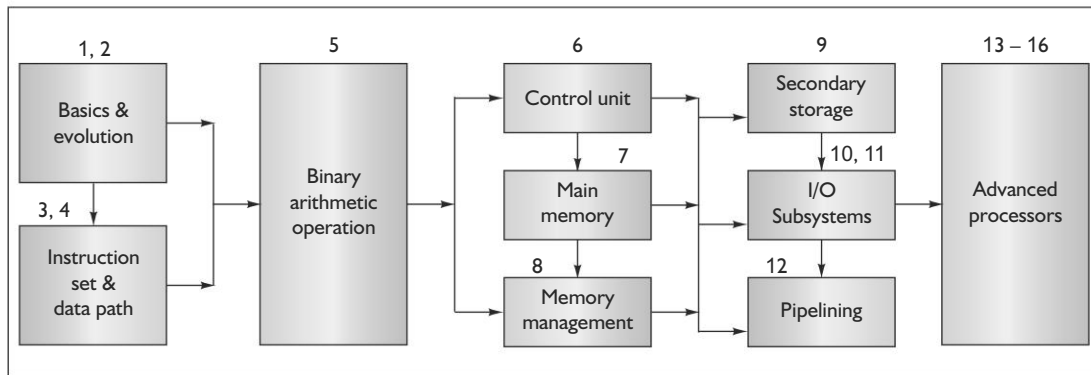
(e) 4.02×10^{14}

Restrict the calculation of magnitude values to the most significant 8 bits.

CHAPTER 5



BINARY ARITHMETIC OPERATIONS



Chapter Flow Diagram

OBJECTIVES

- « Fixed-point arithmetic
- « Serial adder
- « Parallel adder
- « Adder-subtractor
- « Multiplication
- « Booth's algorithm
- « Restoring division
- « Non-restoring division
- « Floating-point arithmetic

KEYWORDS

Overflow	Full-Adder
Ripple Carry	Cascading
Carry Look-Ahead	Multiplicand
Mantissa	Multiplier
Exponent	Partial product
Exponent Overflow	Booth's Algorithm
Exponent Underflow	Dividend
Guard Digits	Divisor
Truncation	Quotient
Rounding	Remainder
Sign Extension	Normalization
Half-Adder	

5.1 Introduction

The datapath contains the ALU, registers, shifters, counters, bus and other hardware paths and circuits that are necessary for moving the data and performing various operations. The datapath components can be configured in many ways. The speed of the datapath unit depends on the algorithms used for the arithmetic operations and the design of the ALU circuits. The knowledge of available arithmetic algorithms is essential to build an efficient datapath. For example, a division instruction can be implemented by two different algorithms: restoring and non-restoring. The restoring division algorithm is straightforward and simple to implement whereas, the non-restoring division algorithm provides a shortcut and yields a faster design. This chapter covers the different algorithms for binary fixed-point and floating-point operations.

5.2 Hardware Components

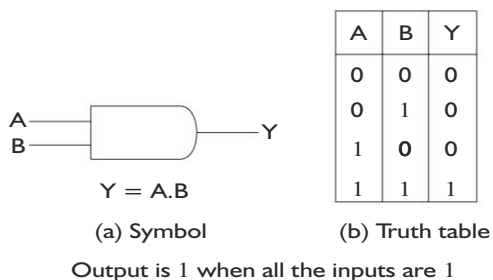
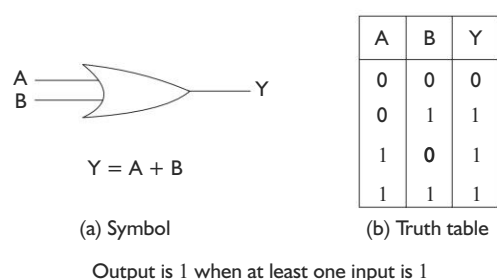
Though our discussion on different arithmetic operation is mostly mathematical, we will be talking in terms of some digital components as well. Most of the readers are familiar with these. A brief summary of certain components that are frequently used in different chapters is presented in Table 5.1. Figures 5.1–5.22 show the symbols and notations used to represent these components. Annexure 4 describes some of the important digital components for the benefit of readers who don't have electronics background.

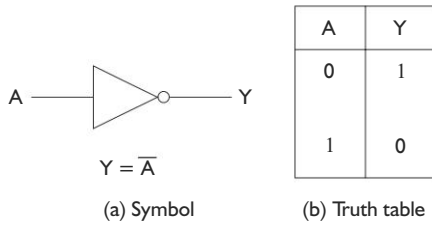
TABLE 5.1 Frequently used hardware components

S. No.	Names of the components	Function	Remarks
1	Gate	A logic circuit that performs a function on inputs and produces an output	Different types are AND, OR, NAND, EXCLUSIVE OR etc.; An IC may contain multiple gates of same type
2	Latch	Locks a signal status; the gate signal controls opening or closing of the latch	Used to grab a condition, before it vanishes, for future requirement

(Contd.)

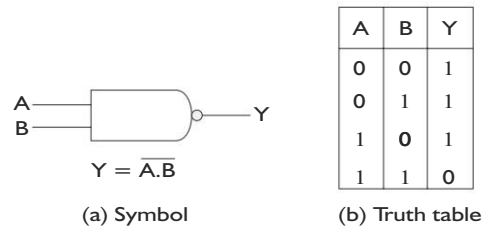
S. No.	Names of the components	Function	Remarks
3	Flip-flop	A one bit storage: 0 or 1; the clock signal triggers the flip-flop and changes state	D and J-K are major types
4	Register	A set of flip-flops with common clock input	Used to store temporary information
5	Shift register	Shifts the inputs left or right as per control inputs	Used in arithmetic operations
6	Counter	Counts the input clock pulses; up or down count	Used in arithmetic and control operations
7	Multiplexer	Selects one out of many inputs as per control inputs	Also known as data selector
8	Demultiplexer	Sends the input to one of many outputs as per control inputs	Useful for serial to parallel conversion
9	Decoder	Identifies the pattern on n inputs and activates the corresponding output out of the 2^n outputs	Used for decoding instruction, command, address etc.
10	Encoder	Selects the active input out of 2^n inputs and identifies it as n-bit output pattern	Functional reverse of a decoder
11	Register file	Multiple registers with a common read/write control	Dual port register file allows simultaneous access on two ports A and B
12	Parity generator	Calculates the parity bit (ODD or EVEN) for the given input bit pattern	Used for error detection: bit drop or pick up
13	Magnitude comparator	Compares two numbers and reports the result: $A < B$, $A = B$ and $A > B$	Useful in arithmetic and logical operations


Fig. 5.1 Two input AND gate

Fig. 5.2 Two input OR gate



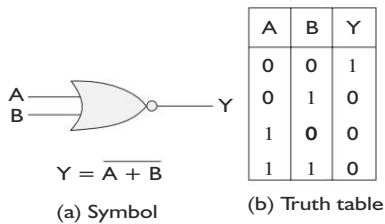
Output is the complement of the input

Fig. 5.3 Inverter (NOT) gate



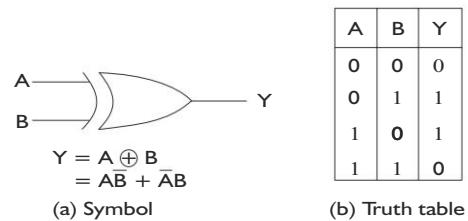
Inversion of AND; output is 1 when at least one input is 0

Fig. 5.4 Two input AND gate



Inversion of OR; output is 1 when all the input are 0

Fig. 5.5 Two input NOR gate



Output is 1 only when the two inputs are complementary

Fig. 5.6 Exclusive OR (XOR) gate

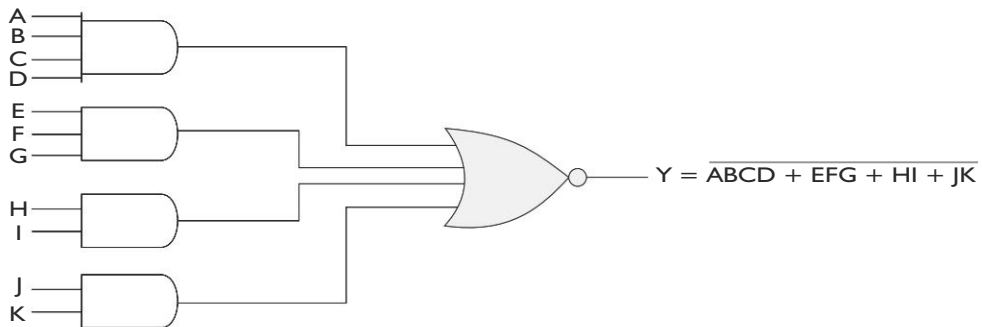


Fig. 5.7 4-3-2-2 Input AOI

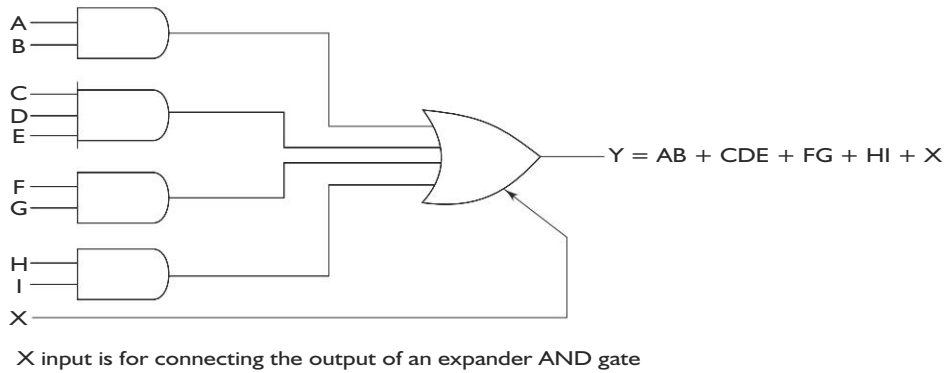


Fig. 5.8 AND-OR with expander (expandable 4 wide AND-OR)

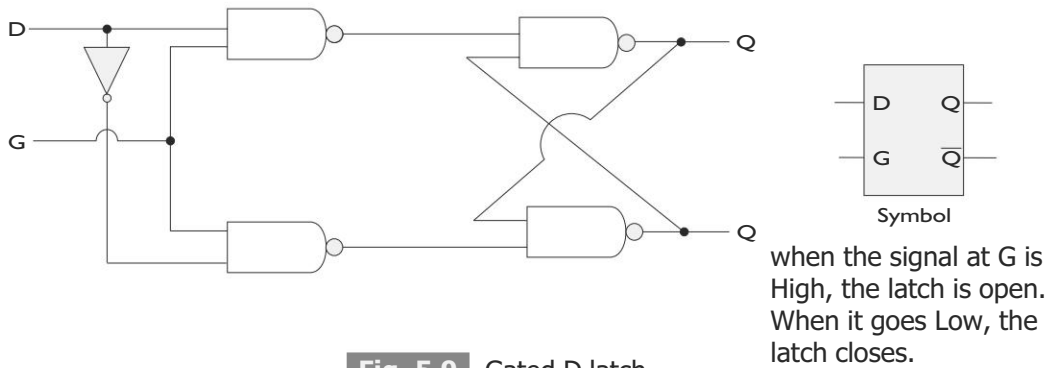
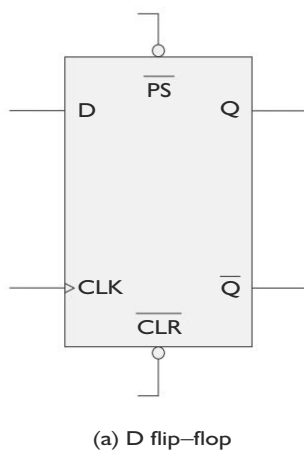


Fig. 5.9 Gated D latch

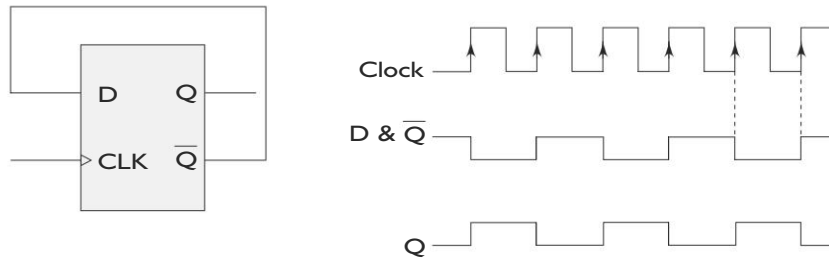


CLK	D	Q	$\bar{Q}$
	0	0	1
	1	1	0
0 or 1	x	Q_p	$\bar{Q}_p$

Q_p and $\bar{Q}_p$ = Previous state

(b) Truth table

Fig. 5.10 D Flip-flop



FF is initially assumed to be in reset state

Fig. 5.11 D Flip-flop as a Frequency Divider

CLK	J	K	Q	$\overline{Q}$
	0	0	No change Q_p $\overline{Q}_p$	
	0	1	0	1
	1	0	1	0
	1	1	$\overline{Q}_p$	Q_p
0 or 1	X	X	No effect	

(a) Symbol

(b) Truth table

X = don't care

Fig. 5.12 J-K Flip-flop

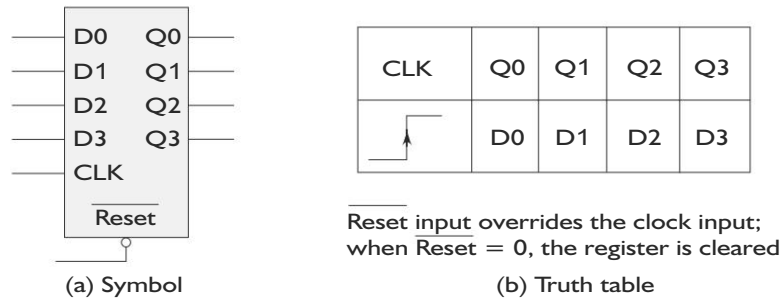


Fig. 5.13 4-bit register

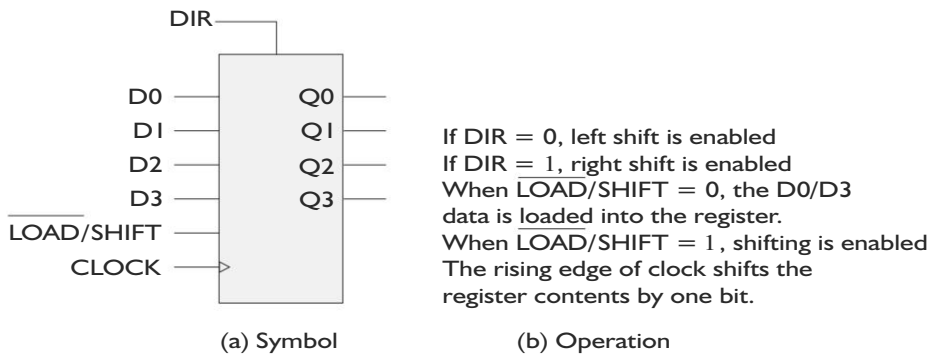


Fig. 5.14 4-bit shift register

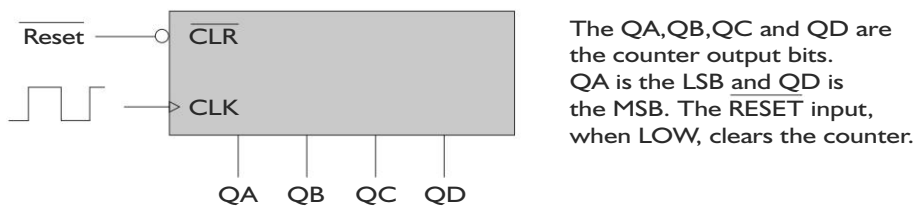


Fig. 5.15 4-bit binary counter

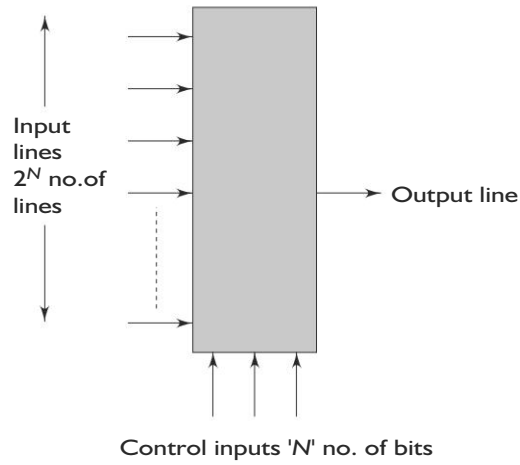


Fig. 5.16 Multiplexer block diagram

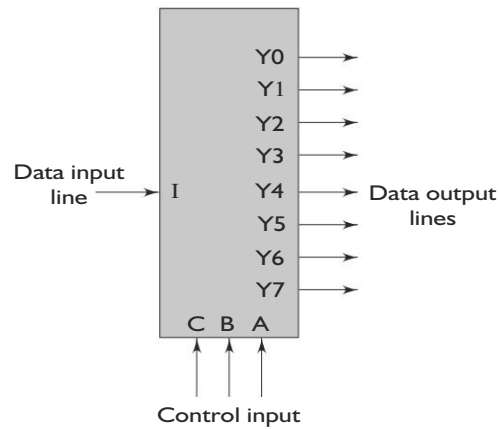


Fig. 5.17 Demultiplexer block diagram

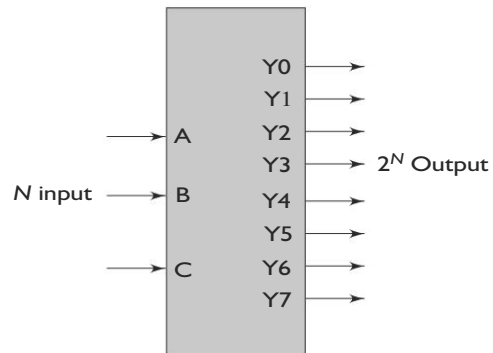
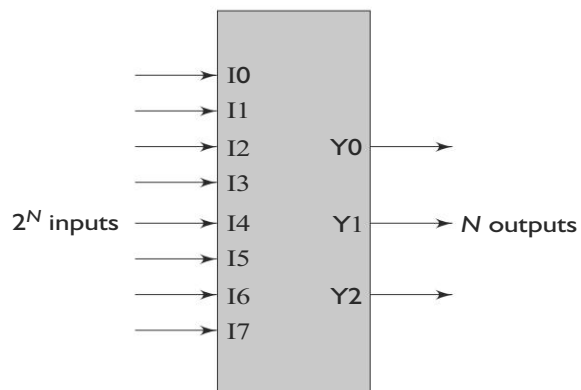
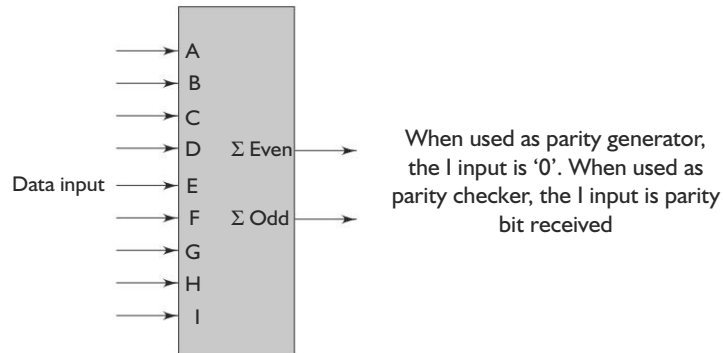


Fig. 5.18 Decoder block diagram

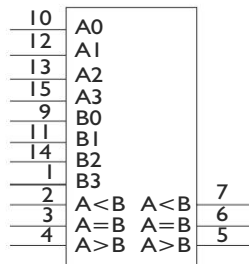


Outputs give encoded pattern

Fig. 5.19 Encoder block diagram


Fig. 5.20 74S280 block diagram

X1 DIGITAL COMPARATOR
(33) 74S85 4 bit comparator



A0 – A3, B0 – B3 – Data inputs; A0 is LSB; A3 is MSB.
2, 3, 4 – Cascading inputs
5, 6, 7 – outputs
Can perform straight binary and straight BCD comparisons.

Function table

Comparing inputs				Cascading inputs			Outputs		
A3, B3	A2, B2	A1, B1	A0, B0	A>B	A<B	A=B	A>B	A<B	A=B
A3>B3	X	X	X	X	X	X	H	L	L
A3<B3	X	X	X	X	X	X	L	H	L
A3=B3	A2>B2	X	X	X	X	X	H	L	L
A3=B3	A2<B2	X	X	X	X	X	L	H	L
A3=B3	A2=B2	A1>B1	X	X	X	X	H	L	L
A3=B3	A2=B2	A1<B1	X	X	X	X	L	H	L
A3=B3	A2=B2	A1=B1	A0>B0	X	X	X	H	L	L
A3=B3	A2=B2	A1=B1	A0<B0	X	X	X	L	H	L
A3=B3	A2=B2	A1=B1	A0=B0	H	L	L	H	L	L
A3=B3	A2=B2	A1=B1	A0=B0	L	H	L	L	H	L
A3=B3	A2=B2	A1=B1	A0=B0	L	L	H	L	L	H
A3=B3	A2=B2	A1=B1	A0=B0	X	X	H	L	L	H
A3=B3	A2=B2	A1=B1	A0=B0	H	H	L	L	L	L
A3=B3	A2=B2	A1=B1	A0=B0	L	L	L	H	H	L

Fig. 5.21

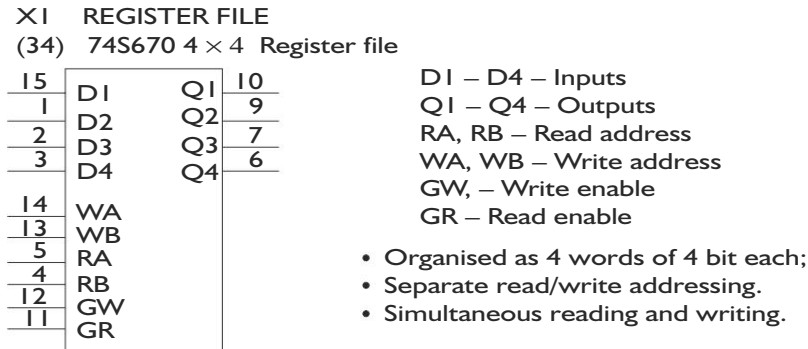


Fig. 5.22

5.3 Fixed-Point Arithmetic

Binary fixed-point arithmetic operations are most important aspect of the ALU. Four types of operations are needed: addition, subtraction, multiplication and division. There are two different strategies for implementing these four operations:

1. Designing the ALU circuits for all the four operations. This approach leads to a costly and expensive ALU.
2. Designing the ALU circuits only for addition and subtraction and performing multiplication and division by software. This approach leads to a slower and cheaper ALU.

5.3.1 Sign Extension

Suppose an n bit number has to be stored in a register or memory location with a longer word length ($\geq n + 1$), then the sign extension rule is followed in filling up the remaining vacant bits (most significant positions). The sign bit is copied (propagated) to all the vacant bits that are to the left of the sign bit. If the number is positive ($S = 0$), the vacant msbs are filled with 0s. If the number is negative ($S = 1$), all vacant msbs are made with 1's. This concept is called *sign extension*. Basically it helps in storing a smaller number of bits in a larger register.

Example 5.1 Show how the following 5 bit signed numbers are stored in 8-bit registers:

(a) 00111 (b) 11001 (c) 00000 (d) 11000 (e) 01010 (f) 11010

(a) 00111 becomes 00000111

(b) 11001 becomes 11111001

(c) 00000 becomes 00000000

(d) 11000 becomes 11111000

(e) 01010 is loaded as 00001010

(f) 11010 is loaded as 11111010

5.3.2 Integer Addition (A + B)

Two types of addition are required in an ALU:

1. *Addition of unsigned numbers*: Operand address calculation is one example where such addition is required.
2. *Addition of signed numbers*: All numeric calculations in a computer generally follow this type of addition since the operands are either positive or negative numbers.

5.3.2.1 Unsigned Numbers Addition

Let us consider the addition of two unsigned fixed-point numbers of n -bits each. The basic requirement for this operation is a set of rules for 1-bit addition. Figure 5.23 gives the rules for 1-bit addition. These rules are same as the truth table contents for an XOR gate shown in Fig 5.6. The carry (C) is generated when both bits are 1. When two numbers of

A	0	0	1	1
B	0	1	0	1
Sum	0	1	1	1 0

carry

Fig. 5.23 Addition rules

multiple bits are added, the carry signal from any bit position should be passed on to the left adjacent (next higher significant) bit position's adder. Figure 5.24(a) shows addition of two 4-bit numbers 1010 (decimal 10) and 0011 (decimal 3). The result is 1101 (decimal 13) which is also a 4-bit number. There is carry out of bit position 1 which is added along with the input bits of (next higher significant) position 2. Consider another example for adding 1100 (decimal 12) with 0101 (decimal 5) as shown in Fig. 5.24(b). The result is 10001 (17) which overflows since it is a 5-bit number. Usually the word length of an addition result is

same as the word lengths of the operands. In our example, the result has to be a 4-bit number and therefore, the result of this addition is 0001. The ALU has a flag called *overflow* which will be set now. It can be sensed by any subsequent instruction. In an unsigned fixed-point addition, if there is a carry from the msb position, it indicates an overflow.

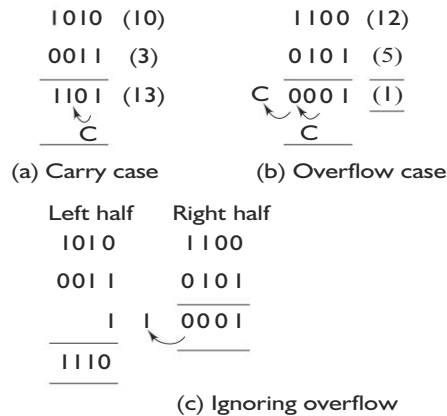


Fig. 5.24 Unsigned binary addition examples

The overflow situation is considered to be a *program exception*. It generates an *interrupt* and the system software takes appropriate action such as terminating the program with an error message.

If a program wants to add two 8-bit numbers in a computer which has a 4-bit adder, the operands have to be split into two halves of 4-bits each. Two add instructions are used. The least significant half is added first. Any carry from the msb now is not an overflow. It has to be carried forward to the other half. Then, the other half is added. Figure 4.24(c) shows how the two 8-bit numbers 10101100 (decimal 172) and 00110101 (decimal 53) are added using a four bit adder to get the result 11100001 (decimal 225). Usually the compiler takes care of converting one addition into two ADD instructions.

5.3.2.1.1 Half-Adder

A half-adder is one that adds two 1-bits but has no provision to include the carry output from the previous bit position. Figure 5.25 shows the block diagram for the half adder. The sum (S) and the carry (C) are given by the following expressions:

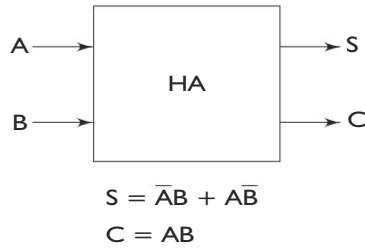


Fig. 5.25 Half-adder block diagram

$$S = \bar{A}B + A\bar{B} \text{ (Here the + sign stands for 'OR' and not for 'plus')}$$

$$= A \oplus B \text{ (Here the } \oplus \text{ sign indicates 'Exclusive OR' function)}$$

$$C = AB$$

Figure 5.26 shows two different forms of logic circuits for the half-adder. (The term *quarter-adder* is used if the adder produces only the sum bit but not the carry bit.) The expression for the sum is the same as that of the exclusive-OR result of the two bits.

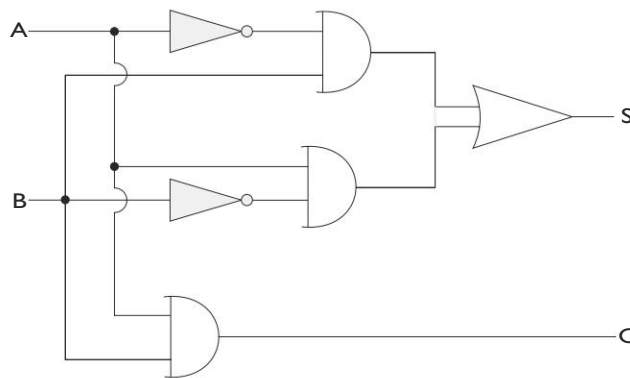


Fig. 5.26(a) Half-adder using AND, OR, NOT gates

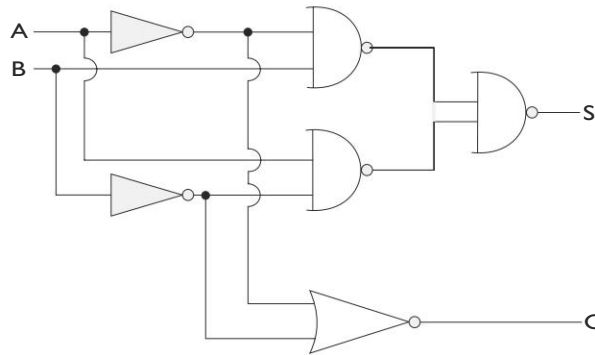
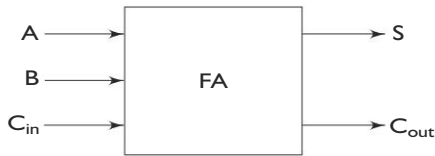


Fig. 5.26(b) Half-adder using NAND, NOR, NOT gates

5.3.2.1.2 Full-Adder

A 1-bit full-adder takes into account the carry from previous bit position also. Basically, it is equivalent to a 1-bit adder of the three numbers. Figure 5.27(a) shows the block diagram for a full-adder and Fig. 5.27(b) shows the truth table. The sum, S , and carry output, C_{out} , are given by the following expressions:



A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Fig. 5.27(a) Full-adder block diagram

Fig. 5.27(b) Full-adder truth table

$$\begin{aligned}
 S &= \bar{A} \bar{B} C_{in} + \bar{A} B \bar{C}_{in} + A \bar{B} \bar{C}_{in} + A B C_{in} \\
 &= A \oplus B \oplus C_{in}
 \end{aligned}$$

$$\begin{aligned}
 C_{out} &= \bar{A} B C_{in} + A \bar{B} C_{in} + A B \bar{C}_{in} + A B C_{in} \\
 &= A C_{in} + A B + B C_{in}
 \end{aligned}$$

The C_{in} is the carry input. A 2-bit full-adder can be constructed from two 1-bit half-adders with some additional logic as shown in Fig. 5.28. This concept can be extended for any

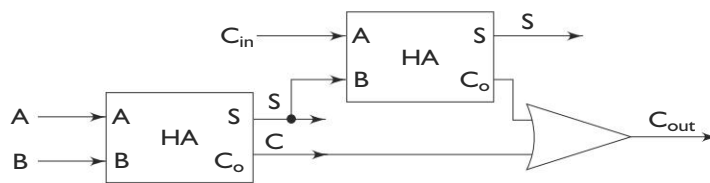


Fig. 5.28 Full-adder using two half-adders

number of bits by adding additional stages of half-adders and circuits. Though this is a modular design approach, there are better methods of designing a full-adder without using half-adder as shown in Fig. 5.29.

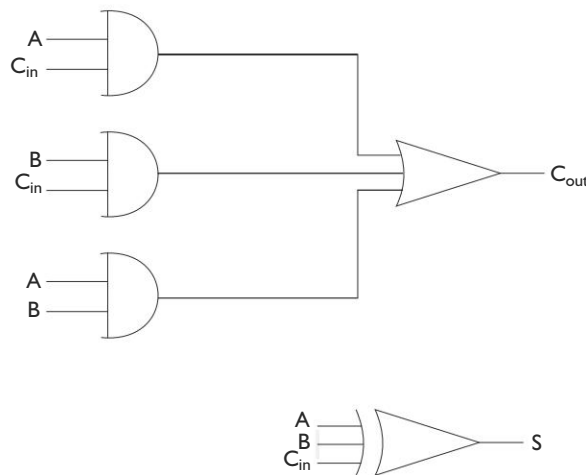


Fig. 5.29 Full-adder circuit without half-adders

5.3.2.1.3 Serial Adder

A serial adder has only a single bit adder. It is used to perform the addition of two numbers sequentially bit by bit starting with lsb. Figure 5.30 shows a serial adder of two n -bit numbers. The operands (A and B) are supplied bit by bit starting with lsbs. Addition of one bit position takes one clock cycle. Thus, for an n -bit serial adder, n clock cycles are needed to complete the addition process and get the result. At each cycle, the carry produced by a

bit position should be remembered in a flip-flop (a 1-bit memory) and it is given as an input during the next (more significant) cycle as carry.

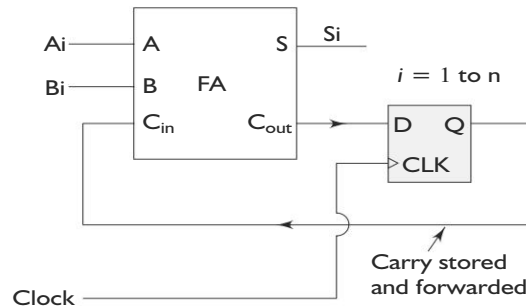


Fig. 5.30 Serial adder

Merits: The circuits for serial adder is less and hence, it is very cheap irrespective of the number of bits to be added.

Demerits: The serial adder is slow since it takes n clock cycles for completing addition of n bit numbers.

5.3.2.1.4 Parallel Adder

A parallel adder is one which has a separate adder circuit for each bit. From the control unit's viewpoint, the parallel adder performs addition of multiple stages (bit positions), simultaneously. However, the internal addition mechanism differs in different types of parallel adders. There are basically two types of parallel adders: *ripple carry adder* and *carry look-ahead adder*. They differ in the way the carry bits are generated. In ripple carry adder, the carry from the LSB position propagates through all the subsequent stages. The carry output of each stage is given as carry input to the next stage. In carry look-ahead adder, carry at each stage is generated without waiting for the carries of previous stages. In other words, the carry at every stage is determined early and directly.

5.3.2.1.5 Ripple Carry Adder

Figure 4.31 shows an n -bit ripple carry adder. It operates as follows:

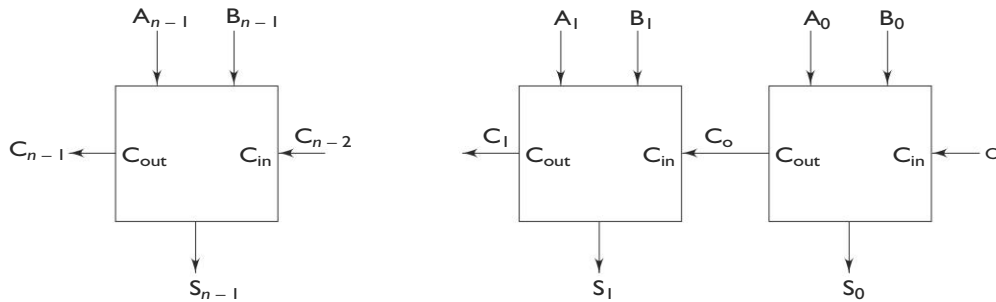


Fig. 5.31 n -bit ripple carry adder

There are n full-adders connected in a cascade mode. Each full-adder's carry output becomes carry input to the next bit full-adder. Each stage performs addition for one of the bits of the two numbers taking into account the carry from previous bit position. Each stage adder produces a sum and a carry bit.

Each stage's carry output is given at the carry input of the next stage adder and propagated by the next stage adder to its output. Each stage adder is a *combinational circuit* in which at any time, the output depends on the current inputs only and has no relevance to earlier inputs. The (sum and carry) outputs of the adder is valid only after sufficient time delay from the instant of supplying the inputs. The C_{in} , the carry input to the lsb stage is kept LOW (0) permanently since it is irrelevant. Sometimes, it has a special use. If the sum has to be incremented by 1, i.e. we need $A + B + 1$, then the C_{in} is made 1.

The delay introduced by the carry decides the addition time. The carry from the lsb position has to propagate through all the bit positions. If t_d is the time delay for each full-adder stage and there are n bits, the maximum propagation delay for an n -bit ripple carry adder is $n \times t_d$. For each stage, the carry is generated with 2 gate delays, whereas the sum is generated with one gate delay. Hence in a 4-bit ripple carry adder, there is 7 gate delays for the sum and 8 gate delays for the carry output. Similarly in the case of a 16-bit ripple carry adder, the number of gate delays between the carry-in, to the LSB, and the carry-out, of the MSB, is $16 \times 2 = 32$.

Advantages: The ripple carry adder is faster than the serial adder.

Disadvantages: The ripple carry adder becomes slow once the number of bits is increased. A carry occurring at the lsb has to propagate through all the subsequent stages. If the length of the operands is large, the time taken for the carry to propagate from lsb to msb stage is more. This is known as ripple carry problem.

Example 5.2 Two unsigned numbers of two bits each are to be added. Which adder is faster: serial adder or ripple carry adder?

In a serial adder, the propagation delay of the flip-flop, t_{FF} also contributes to total delay. If t_d is the delay due to a single stage adder, minimum period of the clock must be $(t_d + t_{FF})$. Hence, minimum time needed by the serial adder is twice this i.e. $2 \times (t_d + t_{FF})$.

In a ripple carry adder, addition time = $2 \times t_d$.

Hence, the ripple carry adder is faster.

Example 5.3 Calculate the time taken by an 8 – bit ripple carry adder.

A stand-alone n -bit ripple carry adder gives out the sum after $(2n)-1$ gate delays whereas the carry out is available after $2n$ gate delays. If any additional gates are used in the paths of the two data inputs (A , B) before they actually reach the adder input, the delay of these gates should be added to the adder delay (addition time).

Hence the 8 – bit ripple carry adder gives the sum after 15 gate delays and the carry out after 16 gate delays.

5.3.2.1.6 Cascading Ripple Carry Adder

By cascading the ripple carry adders, we can construct adders of any length. Fig. 5.32 shows construction of $2n$ -bit adder using two n -bit adders of ripple carry type. Both the modules simultaneously perform addition and then allow the carry propagation from the first module to the next module.

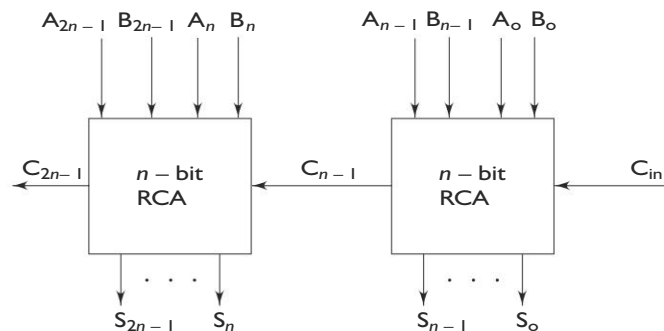


Fig. 5.32 Cascading $2 \times n$ -bit RCA

Example 5.4 Construct a 16-bit parallel adder using 4-bit ripple carry adders.

We need four 4-bit ripple carry adders to form a 16-bit adder. Figure 5.33 shows the interconnections between the four adders.

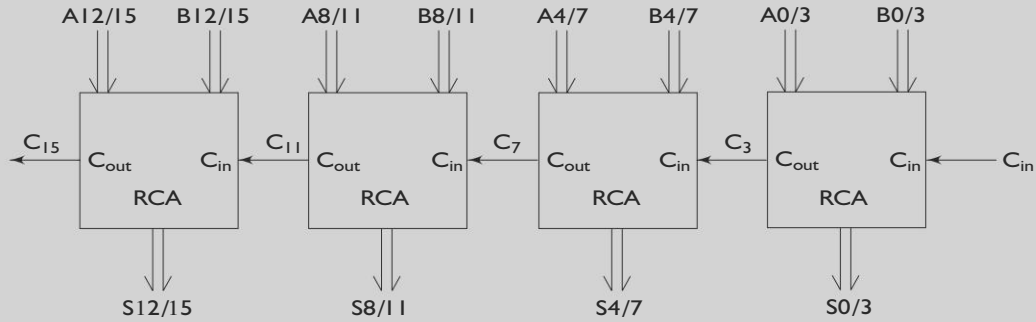


Fig. 5.33 16-bit adder using 4×4 -bit RCA

5.3.2.1.7 Carry Look-Ahead Adder

A carry look-ahead adder is a high speed adder which follows a special strategy for quick carry generation at each stage without waiting for the carries of the previous stages. It involves early determination of the carry input to a stage, without using carry output from the previous stage, by directly determining carry like signals.

Consider an n -bit adder. The carry out of two least significant positions are as follows:

$$C_0 = A_0B_0 ; \quad C_1 = A_1C_0 + A_1B_1 + B_1C_0 = A_1B_1 + C_0 (A_1 + B_1)$$

where C_0 is the carry output from lsb i.e. adder stage 0, and C_1 is the carry output from stage 1. Consider the adder stage i . The carry out from this stage is expressed as:

$$C_i = A_iB_i + (A_i + B_i) C_{i-1}$$

where C_i and C_{i-1} are the carry outputs of stage i and stage $i-1$, respectively.

By carefully analyzing this expression, we find that there are two factors which decide whether there should be a carry bit or not. The first factor depends on the data bits of the current stage and the second factor depends on both the data bit patterns of the current stage and the carry from the previous stage. The first factor can be considered as a carry generation component and the second factor can be considered as a carry propagation. Now, we can rewrite the carry expression as

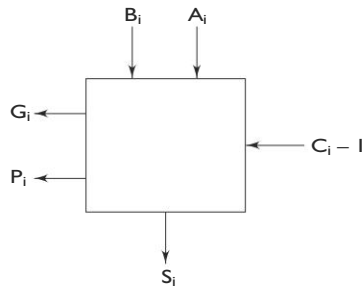
$$C_i = G_i + P_i C_{i-1}$$

where G_i is termed as 'generate' and P_i is termed as 'propagate'. The G_i and P_i are defined as $G_i = A_iB_i$ and $P_i = A_i + B_i$.

Conclusions

1. Stage i generates a carry irrespective of the carry C_{i-1} if A_i and B_i are both 1.
2. Stage i propagates C_{i-1} if A_i or B_i is 1.

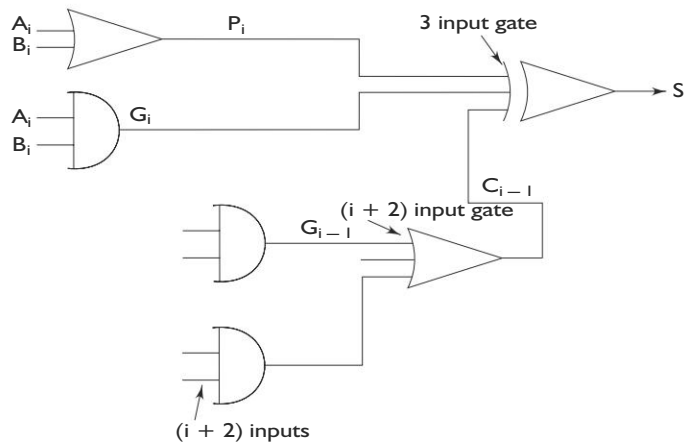
Figure 5.34(a) shows the input and output of each stage and Fig. 5.34(b) gives the logic diagram. The expression for the carry C_i can be expanded as follows:



(a) Symbol

Fig. 5.34(a) One stage of carry look-ahead adder

$$\begin{aligned} C_i &= G_i + P_i C_{i-1} \\ &= G_i + P_i (G_{i-1} + P_{i-1} C_{i-2}) \end{aligned}$$



(b) Logic

Fig. 5.34(b) One stage of carry look-ahead adder

Thus, C_i is a sum of the products of P and G outputs of the preceding stages. Extending this statement for all the carries, it is obvious that all the carries in a parallel adder can be derived directly from the input data bit patterns without actually waiting for the carry status of preceding bit positions. This means that every stage can perform addition independent of other stages. This increases the speed of addition but several gates with multiple inputs are needed as the adder length (n) increases.

Fig. 5.35(a) gives a block diagram of a 4-bit carry look-ahead adder and Fig. 5.35(b) shows the symbol. The carries are derived as follows:

$$C_0 = G_0 + P_0 C_{in}$$

$$C_1 = G_1 + P_1 G_0 + P_1 P_0 C_{in}$$

$$C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in}$$

$$C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in}$$

The G_0 and P_0 are 'super generate' and 'super propagate' signals which can be used for cascading.

C_{in} is the carry input to lsb stage which is normally 0. It is easy to calculate the propagation delay of the 4-bit adder as $4 \times d$ where d is the average gate delay. The P_i and G_i signals are generated after one gate delay from the application of the inputs A, B and C_{in} , as seen in Fig. 5.34 (b). The $C_{(i-1)}$ signal is generated by the AND-OR gates with an additional delay of two gates. The XOR gate adds one more gate delay. Thus the total delay is of four gates for the sum and three gates for the carry. The same holds good for any number of bits because

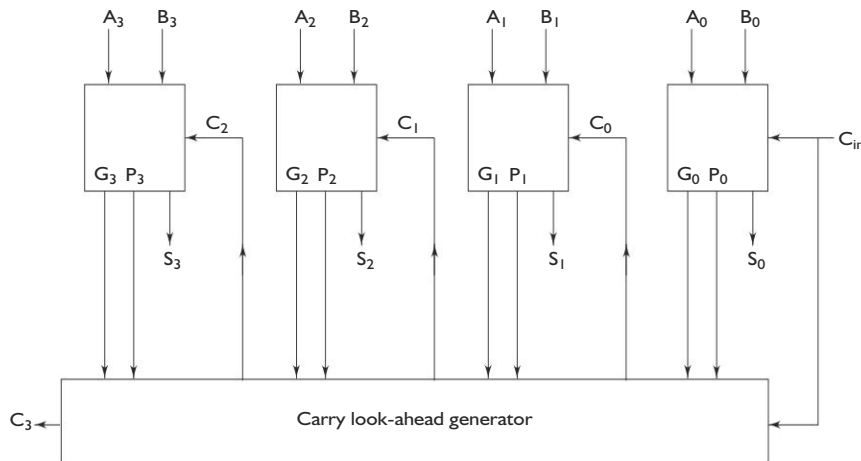


Fig. 5.35(a) 4-bit carry look-ahead adder

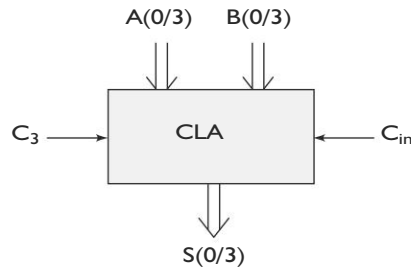


Fig. 5.35(b) Symbol for 4-bit carry look ahead adder

the adder delay does not depend on word length n in the case of carry look-ahead adder. It depends on the number of levels of gates used to generate the sum and the carry bits. All the carry bits are generated simultaneously.

Example 5.5 Calculate the delay for an 8-bit carry look-ahead adder. Assume that the average gate delay is 5 ns.

For carry look-ahead adder, the adder delay is four gates delay i.e. $4 \times 5 = 20$ ns.

Example 5.6 Following two 4-bit numbers are added by a 4-bit carry look-ahead adder. Determine the carryout C_3 .

$$A = 0011$$

$$B = 1011$$

The relevant carry expression is $C_i = G_i + P_i C_{i-1}$ where the generate $G_i = A_i B_i$ and the 'propagate' $P_i = A_i + B_i$.

$$C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in}$$

Let us find out the various G s and P s.

$$A = 0011$$

$$B = 1011$$

$$G_1 = 0011$$

$$P_1 = 1011$$

$$\therefore G_3 = 0, G_2 = 1, G_1 = 1, G_0 = 1$$

$$P_3 = 1, P_2 = 0, P_1 = 1, P_0 = 1$$

$$\begin{aligned} \therefore C_3 &= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in} \\ &= 0 + 0 + 0 + 0 + 0 = 0 \end{aligned}$$

Example 5.7 What is carry out C_3 if the following two 4-bit numbers are added:

$$A = 0011$$

$$B = 1110$$

Let us find out the various G s and P s.

$$A = 0011$$

$$B = 1110$$

$$G_1 = 0010$$

$$P_1 = 1111$$

$$G_3 = 0, G_2 = 0, G_1 = 1, G_0 = 0$$

$$P_3 = 1, P_2 = 1, P_1 = 1, P_0 = 1$$

$$\begin{aligned} C_3 &= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_{in} \\ &= 0 + 0 + 1 + 0 + 0 = 1 \end{aligned}$$

5.3.2.1.8 Building Long Adders

For designing a long adder to add two 32-bit numbers, the carry output of the last stage, C_{31} is generated by a 33 input OR gate. One of the inputs ($P_{31}P_{30} \dots P_0C_{in}$) to this OR gate itself is generated by a 33 input AND gate. Such large gates are not practically manufactured since their requirements are rare. Hence, to build an adder for a long word length, a multi-level design is followed by cascading modules of short adders of 4-bits or 8-bits. Again, we have two different approaches for building a 16-bit adder as illustrated below:

1. Use a four 4-bit carry look-ahead adder modules and cascade them in a ripple carry fashion as shown in Fig. 5.36. This adder will be faster than a 16-bit ripple carry adder.
2. Use a four carry look-ahead adder modules and cascade them in carry look-ahead fashion as shown in Fig. 5.37. In this design, two levels of carry look ahead is performed. In the first level, the four 4-bit CLAs generate 'super propagate' and 'super generate' signals which are given as inputs to the second level carry look ahead logic. The carry in C_{in} is input to both the second level and first-level carry look-ahead logic (to the first one). The second level CLA generates four carry output signals Cs_0 , Cs_1 , Cs_2 and Cs_3 . The Cs_3 is the final carry output of the adder and hence named as C_{15} . The Cs_0 to Cs_2 are given as carry in to the last three CLAs in first level.

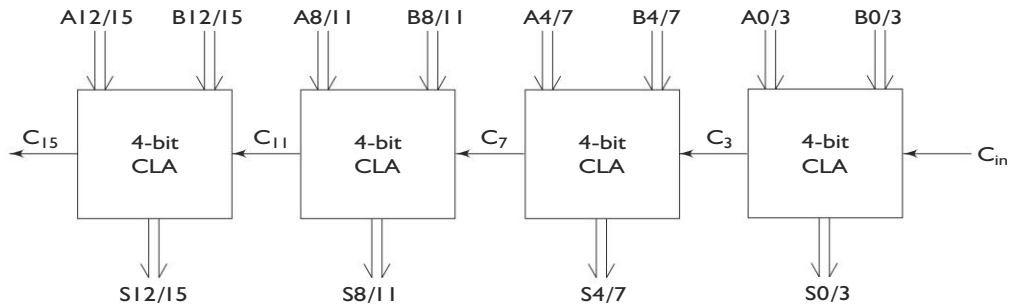


Fig. 5.36 Cascading $4 \times$ CLAs in ripple carry fashion

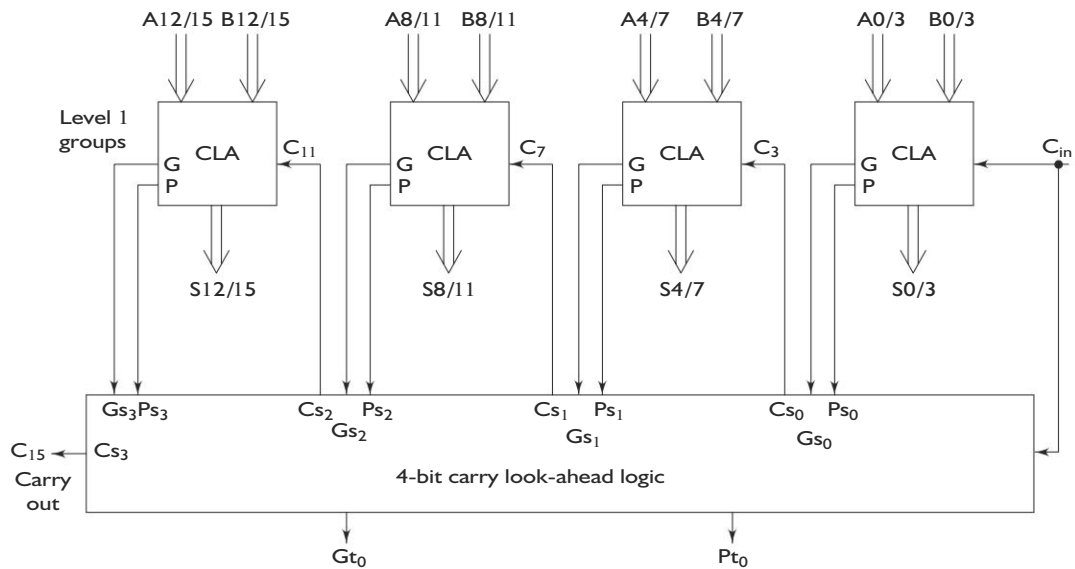


Fig. 5.37 16-bit adder using $4 \times$ 4-bit CLAs in carry look-ahead fashion

The 'super propagate' signals are defined by the following equations.

$$P_{s0} = P_3 \cdot P_2 \cdot P_1 \cdot P_0$$

$$P_{s1} = P_7 \cdot P_6 \cdot P_5 \cdot P_4$$

$$P_{s2} = P_{11} \cdot P_{10} \cdot P_9 \cdot P_8$$

$$P_{s3} = P_{15} \cdot P_{14} \cdot P_{13} \cdot P_{12}$$

It can be easily observed that for a given group (of 4 bits) in the first level CLA, the 'super propagate' signal is 'true' only if each bit in the group propagates a carry.

The 'super generate' signals are defined by the following equations:

$$Gs_0 = G_3 + P_3 \cdot G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0$$

$$Gs_1 = G_7 + P_7 G_6 + P_7 P_6 G_5 + P_7 P_6 P_5 G_4$$

$$Gs_2 = G_{11} + P_{11} G_{10} + P_{11} P_{10} G_9 + P_{11} P_{10} P_9 G_8$$

$$Gs_3 = G_{15} + P_{15} G_{14} + P_{15} P_{14} G_{13} + P_{15} P_{14} P_{13} G_{12}$$

$$C_{15} = G_3 + (P_3 \cdot G_2) + (P_3 \cdot P_2 \cdot G_1) + (P_3 \cdot P_2 \cdot P_1 \cdot G_0) + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_{in}$$

The G_{t0} and P_{t0} are next level 'super generate' and 'super propagate' outputs which can be used if there are three levels of cascading. An A64-bit adder can be constructed by three levels of cascading. We will be needing four sets of 16-bit adders.

Example 5.8 Determine the G & P values for the addition of following two numbers in carry look-ahead adder. Also determine the carryout bit from the MSB position.

$$A = 1110\ 0101\ 1110\ 1011$$

$$B = 0001\ 1010\ 0011\ 0011$$

Let us first determine G_i and P_i using the expression $G_i = A_i B_i$ and $P_i = A_i + B_i$

$$A = 1110\ 0101\ 1110\ 1010$$

$$B = 0001\ 1010\ 0011\ 0011$$

$$G_i = 0000\ 0000\ 0010\ 0010$$

$$P_i = 1111\ 1111\ 1111\ 1011$$

Next, let us determine the super propagates

$$Ps_0 = P_3 \cdot P_2 \cdot P_1 \cdot P_0 = 1 \cdot 0 \cdot 1 \cdot 1 = 0$$

$$Ps_1 = P_7 \cdot P_6 \cdot P_5 \cdot P_4 = 1 \cdot 1 \cdot 1 \cdot 1 = 1$$

$$Ps_2 = P_{11} \cdot P_{10} \cdot P_9 \cdot P_8 = 1 \cdot 1 \cdot 1 \cdot 1 = 1$$

$$Ps_3 = P_{15} \cdot P_{14} \cdot P_{13} \cdot P_{12} = 1 \cdot 1 \cdot 1 \cdot 1 = 1$$

Next, let us determine the super generates.

$$Gs_0 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0$$

$$= 0 + (1 \cdot 0) + (1 \cdot 0 \cdot 1) + (1 \cdot 0 \cdot 1 \cdot 0)$$

$$= 0 + 0 + 0 + 0 = 0$$

(Contd.)

$$\begin{aligned}
Gs_1 &= G_7 + P_7 G_6 + P_7 P_6 G_5 + P_7 P_6 P_5 G_4 \\
&= 0 + (1 \cdot 0) + (1 \cdot 1 \cdot 1) + (1 \cdot 1 \cdot 1 \cdot 0) = 0 + 0 + 1 + 0 = 1 \\
Gs_2 &= G_{11} + P_{11} G_{10} + P_{11} P_{10} G_9 + P_{11} P_{10} P_9 G_8 \\
&= 0 + (1 \cdot 0) + (1 \cdot 1 \cdot 0) + (1 \cdot 1 \cdot 1 \cdot 0) = 0 + 0 + 0 + 0 = 0 \\
Gs_3 &= G_{15} + P_{15} G_{14} + P_{15} P_{14} G_{13} + P_{15} P_{14} P_{13} G_{12} \\
&= 0 + (1 \cdot 0) + (1 \cdot 1 \cdot 0) + (1 \cdot 1 \cdot 1 \cdot 0) = 0 + 0 + 0 + 0 = 0 \\
C_{15} &= Gs_3 + (Ps_3 \cdot Gs_2) + (Ps_3 \cdot Ps_2 \cdot Gs_1) \\
&\quad + (Ps_3 \cdot Ps_2 \cdot Ps_1 \cdot Gs_0) \\
&\quad + (Ps_3 \cdot Ps_2 \cdot Ps_1 \cdot Ps_0 \cdot C_{in}) \\
&= 0 + (1 \cdot 0) + (1 \cdot 1 \cdot 1) + (1 \cdot 1 \cdot 1 \cdot 0) + (1 \cdot 1 \cdot 1 \cdot 0 \cdot 0) \\
&= 0 + 0 + 1 + 0 + 0 = 1
\end{aligned}$$

Hence a carry out is generated when adding these two numbers.

5.3.2.2 Signed Two's Complement Addition

The addition of signed numbers can be done using 1's complement representation but the addition of 2's complement numbers is more common in computers because of its direct suitability for implementation without any complication. Before listing the rules of 2's complement addition, let us consider some examples. Figure 5.38 illustrates the addition of two numbers ($A + B$) in six different sets. We are assuming that we have a five bit ALU including the sign bit. As per the 2's complement representation, positive numbers are represented without any change (in true binary) with a sign bit of 0. The negative numbers have sign bit of 1 and their magnitudes are converted into 2's complement form. All the bits including the sign bits take part in addition process. In the case of Fig. 5.38(a), the sign bits are different and there are carry out of the msb and sign bit positions. The result is correct since $(+7) + (-3) = (+4)$ and it is a positive number. In the case of 5.38(b), signs of the numbers are different. There is no carry from msb or sign bit position. This case is a simple one $(-6) + (+5) = (-1)$. The result is negative and in 2's complement form. In the case of 5.38(c), both the numbers are positive. There is a carry from msb position but no carry from the sign bit position. The sign bit of the result is 1 indicating a negative value which is incorrect. This case is a classic example of the overflow because $(+9) + (+8) = +17$, whereas the maximum positive numbers which can be represented with 4 bits is 15. Hence in this case, the result is not a true sum. In the case of 5.38(d), two negative

numbers are added. There are carry both from msb and sign positions. The result is negative and valid but in 2's complement form. $(-9) + (-3) = (-12)$. In the case of 5.38(e), both

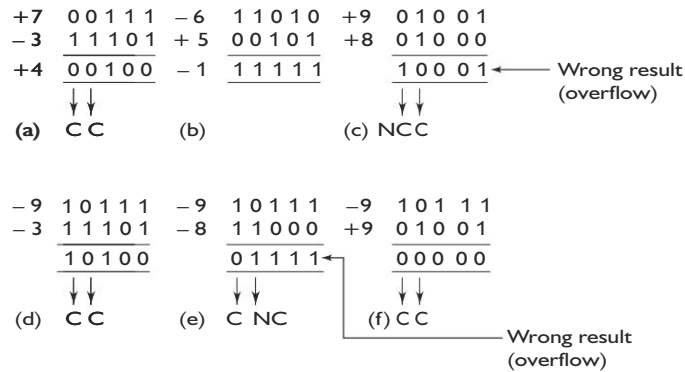


Fig. 5.38 2's complement addition examples

numbers are negative but the result shown is positive. There is no carry from the msb position but the sign bit position has a carry. This case is a overflow example since $(-9) + (-8) = -17$, and the maximum negative numbers that can be represented with a 4 bit magnitude is -16 . In case of 5.38(f), two numbers of equal magnitude but opposite sign are added. There are carries from both msb and sign bit positions. The result is shown as positive 0. $(-9) + (+9) = (+0)$.

Conclusions on 2's Complement Addition

1. Addition of two numbers with different signs does not cause overflow.
2. If the signs of the two numbers are same, there may be overflow. If the sign of the result is different, it indicates overflow. If the result's sign is also the same as the numbers, there is no overflow. The overflow is indicated if the carry status from the msb position and the sign bit position differs. In other words there is no overflow if there are 'carries' from both these positions or 'no carry' from both.

5.3.3 Integer Subtraction

Figure 5.39 gives the rules for 1-bit subtraction.

$$\begin{aligned} 0 - 0 &= 0 \\ 0 - 1 &= 1, \text{ with borrow 1 from adjacent MSB} \\ 1 - 0 &= 1 \\ 1 - 1 &= 0 \end{aligned}$$

Fig. 5.39 Rules of Binary subtraction

Subtraction of fixed-point numbers ($A - B$) is done by adding 2's complement of B to A . Hence, the first step of subtraction is converting the second operand into its 2's complement form. Then, the remaining steps are same as the addition process. An adder can be easily modified to do 2's complement subtraction. Let us consider $(A) - (B)$. It can be replaced by $(A) + (B_{2c})$ where B_{2c} is 2's complement of B . The 2's complement of B is obtained by adding 1 to the 1's complement of B . An easy method for converting a number into 1's complement is by doing Exclusive OR addition of that number with all 1's. Hence, the procedure for subtraction ($A - B$) is as follows:

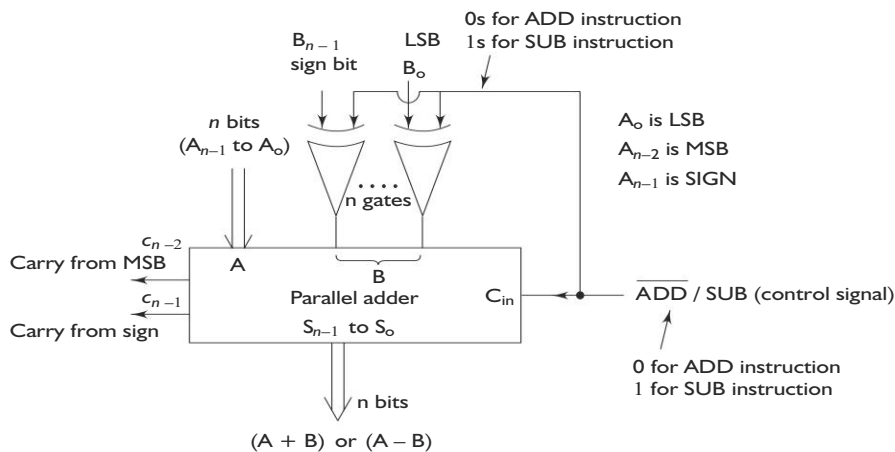
1. Do Exclusive OR of B with 1111..1.
2. Add the above result (1's complement) with A along with a 1 to the carry-in position (lsb). This is equivalent to adding A with B_{2c} .

Figure 5.40 shows some examples for 2's complement subtraction. Overflow occurs in subtraction in two cases:

1. When a negative number is subtracted from a positive number and the result is negative
2. When a positive number is subtracted from a negative number and the result is positive

Figure 5.41 shows a single circuit serving as an adder cum subtractor for 2's complement numbers. By decoding the opcode, the control unit knows whether the current instruction is ADD or SUB instruction and accordingly configures the circuit either for addition or subtraction mode. For ADD instruction, the $\overline{ADD}/SUB$ signal is LOW and for SUB instruction, the $\overline{ADD}/SUB$ signal is high. Since this signal is given to C_{in} input of the adder, it is used for adding 1 to 1's complement of B . The same signal performs Exclusive OR of all 1's to B for SUB instruction. For an ADD instruction, the circuit shows that Exclusive OR of all 0's are done to B . This has no meaning because B is unaffected. (Exclusive OR of a bit to 0 is same as that bit.) Hence, for add operation it simply transfers the contents from the input of Exclusive OR gates to the output.

$ \begin{array}{r} +7 \quad 00111 \\ (-) \\ -3 \quad 11101 \rightarrow 2^5C \\ \hline \end{array} $ (a) $ \begin{array}{r} -6 \quad 11010 \\ (-) \\ -5 \quad 11011 \rightarrow 2^5C \\ \hline \end{array} $ (b) $ \begin{array}{r} -6 \quad 11010 \\ (-) \\ +5 \quad 00101 \rightarrow 2^5C \\ \hline \end{array} $ (c)	$ \begin{array}{r} 00111 \\ (+) \\ 00011 \\ \hline 01010 \quad (+10) \end{array} $ $ \begin{array}{r} 11010 \\ (+) \\ 00101 \\ \hline 11111 \quad (-1) \end{array} $ $ \begin{array}{r} 11010 \\ 11011 \\ \hline 10101 \quad (-11) \end{array} $
---	---

Fig. 5.40 2's complement subtraction examples**Fig. 5.41** Adder/subtractor circuit

5.3.4 Integer Multiplication

Figure 5.42 gives the rules for binary multiplication.

$$\begin{aligned}
 0 \times 0 &= 0 \\
 0 \times 1 &= 0 \\
 1 \times 0 &= 0 \\
 1 \times 1 &= 1
 \end{aligned}$$

Fig. 5.42 Rules of Binary multiplication

Integer multiplication is more complex than addition. It takes more time than addition as multiplication depends on bit pattern of the operands ($A \times B$). There are intelligent algorithms which further shorten the time by shortcut techniques. The length of the product is $2n$ bits when the multiplier and multiplicand are each n bits.

5.3.4.1 Multiplication of Unsigned Numbers

Theoretically, multiplication can be done by repeated addition. Consider multiplication $A \times B$ where B is the multiplier and A is the multiplicand. If we add A to itself B times, the sum will be the product of $A \times B$. This is never done in practical computers since it is a very slow process. Another simple and practically possible method is multiplying A by bit-by-bit of B and adding all the partial products. This method is similar to manual multiplication by *paper and pencil* method as shown in Fig. 5.43 for 1010 (decimal 10) by 1101 (decimal 13). The result is 10000010 (decimal 130), which is obtained by adding the four partial products (1010, 0000, 1010 and 1010) with appropriate shifts as shown in figure.

$$\begin{array}{r}
 1010 \times 1101 \\
 \hline
 1010 \\
 0000 \\
 1010 \\
 1010 \\
 \hline
 10000010
 \end{array}$$

Fig. 5.43 Multiplication—manual method

This method involves generating n *intermediate products* (where n is the number of bits in the multiplier) and then adding them properly taking into account the weight of each bit while moving from lsb to msb. The following steps are used in manual multiplication:

1. *Start with lsb of the multiplier.* If it is 1, the multiplicand itself is the current partial product. If the lsb is 0, put 0s as the current partial product.
2. *Analyse next bit of the multiplier* and generate the partial product as per the rule followed in step 1. This new partial product is written with one bit left shift compared to the previous one. Repeat this step, till all bits of the multiplier are covered.
3. *Add all the partial products* written in step 2 to get the final product.

The previous procedure is slightly modified while implementing in a computer so that multiplication is done faster by the hardware using the parallel binary adder. The partial product is added immediately after each cycle instead of adding at the end so that, we do not store the partial products. The intermediate product is shifted right by one bit after each cycle so that, the multiplicand can be added straight in the next cycle. Figure 5.44 shows the block diagram of multiplication operation. Figure 5.45 shows the algorithm. Initially, the accumulator is cleared and the multiplier and the multiplicand are available in MR and

MD registers. The carry out from the adder is registered in C flag. The C flag also participates in right shift along with the accumulator and the multiplier register. Figure 5.46 shows the status of different registers after each bit analysis.

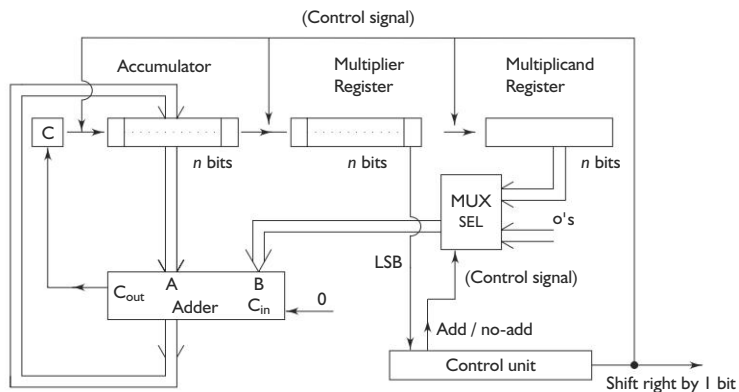


Fig. 5.44 Block diagram—unsigned numbers multiplication

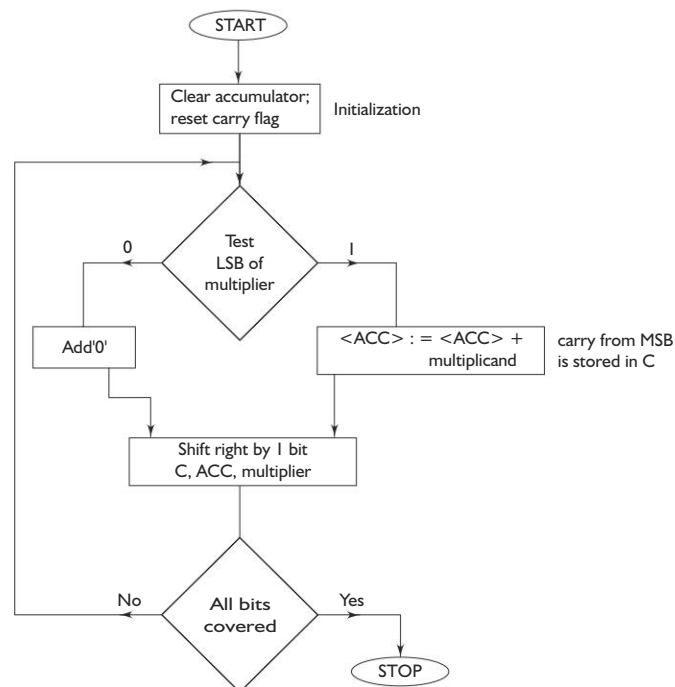


Fig. 5.45 Algorithm for multiplication

	C	ACC	Multiplier MR	Multiplicand MD
Initial	0	0 0 0 0	1 0 0 1	1 0 1 1
T	0	1 0 1 1		MR = 1; Add MD
①	0	1 0 1 1	1 0 0 1	Shift right
	0	0 1 0 1	1 1 0 0	After shift
				MR = 0; No add, shift
②	0	0 0 1 0	1 1 1 0	After shift MR = 0; No add, shift
③	0	0 0 0 1	0 1 1 1	After shift MR = 1; add, MD
④	0	1 0 1 1		
	0	1 1 0 0	0 1 1 1	Shift right
	0	0 1 1 0	0 0 1 1	After shift

Fig. 5.46 Multiplication: status of registers

5.3.4.2 Multiplication of Signed Integers

Multiplication of the numbers in signed magnitude form is easy. Positive numbers can be multiplied in the same way as unsigned numbers. If the signs of the multiplier and multiplicand are different, the sign of the product is negative. The magnitudes are multiplied in the same way as unsigned numbers. If both numbers are negative, the sign of the result is positive. Otherwise, this case is same as positive numbers. The sign requirements are met by the general rule: the sign of the product is Exclusive OR of the signs of the two numbers.

5.3.4.3 2's Complement Multiplication

If one of the negative operand is in 2's complement form, the previous method cannot be directly used. The number has to be converted into sign magnitude form before multiplication and the product has to be reconverted into 2's complement form. Though it works well, it increases the total time taken for multiplication. There are several methods for 2's complement multiplication. Booth's algorithm is a popular method for direct 2's complement multiplication. It also speeds up the multiplication process by analyzing multiple bits of multiplier at a time.

5.3.4.4 Booth's Algorithm

The Booth's algorithm handles both positive and negative (2's complement) operands without need for any correction in the procedure or result. When the multiplier has a stream of

1's, the number of additions required are minimized. This speeds up the multiplication operation. The exact amount of time reduction depends on the bit pattern in the multiplier. The basic principle followed in Booth's algorithm in this: multiplication is nothing but addition of properly shifted multiplicand patterns.

Booth's algorithm involves recoding the multiplier first. In the recoded format, each bit in the multiplier can take any of three values: 0, 1 and -1 . Suppose we want to multiply a number by 01110 (decimal 14). This number can be considered as the difference between 10000 (decimal 16) and 00010 (decimal 2). The multiplication by 01110 can be achieved by summing up the following two products:

- (a) 2^4 times the multiplicand ($2^4 = 16$).
- (b) 2's complement of 2^1 times the multiplicand ($2^1 = 2$).

Any sequence of 1's in a binary number can be broken into the difference of two binary numbers. Hence the multiplication by a string of 1's can be replaced by simpler operations: adding the multiplier, shifting the partial product by appropriate places and then finally subtracting the multiplier. Essentially, a string of 1's in the multiplier is replaced with first a subtraction on seeing a first 1, and then an addition later, on seeing the bit next to the last 1. A string of 0's does not require any addition or subtraction as in normal algorithms.

In a standard multiplication, three additions are required due to the string of three 1's. This can be replaced by one addition and one subtraction. The above requirement is identified by recoding of the multiplier 01110 using following rules which are summarized in Table 5.2.

TABLE 5.2 BOOTH's recoding rules

A_i	A_{i-1}	Recoding for A_i	Remarks on multiplier bit pattern	Action
0	0	0	Stream of 0s	Nothing
1	0	$\bar{1}$	Start of stream of 1s	Subtraction
1	1	0	Stream of 1s	Nothing
0	1	1	End of stream of 1s	Addition

1. Start from lsb (right most bit). Check each bit one by one. Do not disturb 0s and continue till you see the first 1.
2. Change the first 1 as $\bar{1}$.
3. Skip all succeeding 1s (recode them as 0s) till you see a 0. Change this 0 as 1.
4. Continue to look for next 1 without disturbing 0s and proceed using rules 2 and 3.

$$\text{Original number} = 0\ 1\ 1\ 1\ 0 = (0 + 8 + 4 + 2 + 0) = 14$$

$$\text{Recoded form} = 1\ 0\ 0\ \bar{1}\ 0 = (16 + 0 + 0 - 2 + 0) = 14$$

From the previous paragraphs, we confirm that the multiplication by 01110 can be replaced by the addition of products of two simple multiplications. Nothing has to be done except shift operation while we are dealing with 0's in a binary multiplier. This method can be extended to any number of stream of 1's in the multiplier.

Example 5.9 Convert the following numbers into Booth's recoded format:

(a) 01010101

(b) 11001100

(c) 00111011

(a) Original number = 0 1 0 1 0 1 0 1

Recoding = 1 $\bar{1}$ 1 $\bar{1}$ 1 $\bar{1}$ 1 $\bar{1}$

(b) Original number = 1 1 0 0 1 1 0 0

Recoding = 0 $\bar{1}$ 0 1 0 $\bar{1}$ 0 0

(c) Original number = 0 0 1 1 1 0 1 1

Recoding = 0 1 0 0 $\bar{1}$ 1 0 $\bar{1}$

5.3.4.5 Multiplication Using Booth's Algorithm

The implementation of Booth's algorithm for multiplication appears complex but can be made easy if Table 5.2 is followed carefully. The Booth's algorithm performs an addition when it finds the first digit of a stream of 1s (011..) and performs a subtraction when it finds the end of the stream (..10). While analyzing the lsb, it should be assumed that a 0 exists before lsb. Figure 5.47 shows the block diagram of the hardware configuration for Booth's algorithm. Figure 5.48 gives the flow chart for the implementation of Booth's algorithm. A flip-flop (Q_{0-1}) is used to the right of lsb of the multiplier. Initially, it is reset. Subsequently, it receives the lsb of the multiplier when the multiplier is shifted right. The multiplier and multiplicand are loaded as shown in Fig. 5.49. The accumulator is cleared initially.

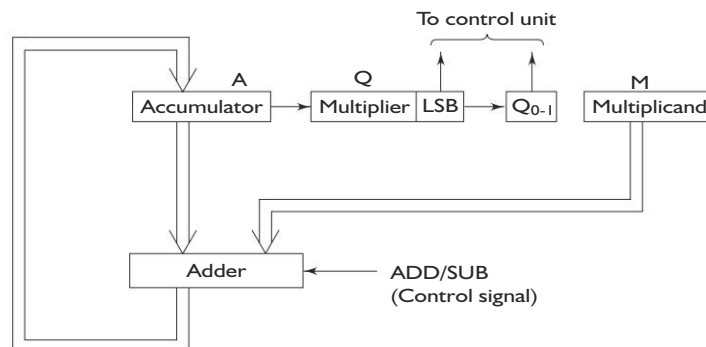
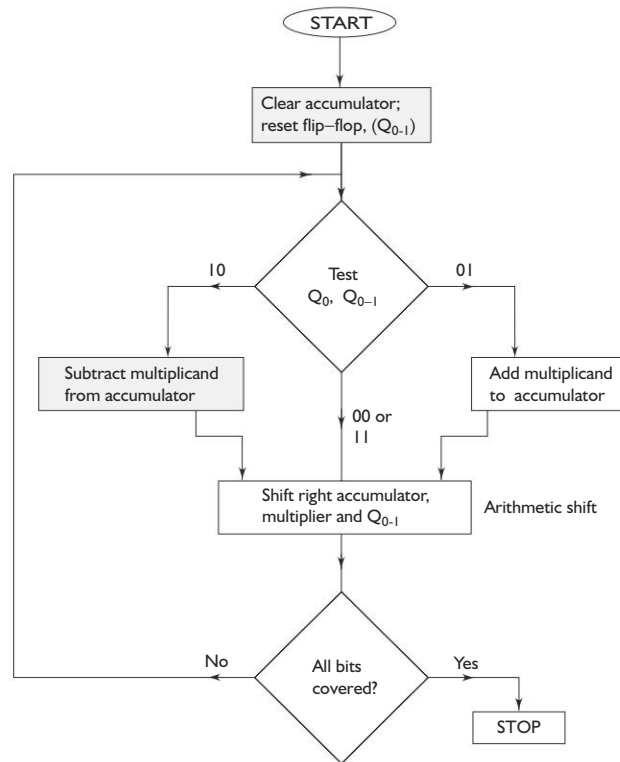
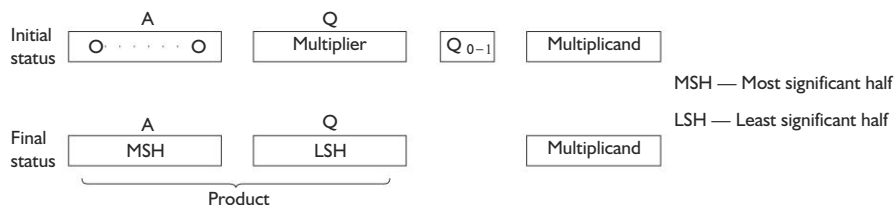


Fig. 5.47 Block diagram for Booth's algorithm

**Fig. 5.48** Booth's algorithm**Fig. 5.49** Registers status-booth's algorithm

Multiplier bits are considered one at a time together with the previous bit.

1. If current and previous bits are same (00 or 11), neither addition nor subtraction is done. The contents of accumulator (partial product) and multiplier are shifted right by one bit. The lsb of the accumulator enters the msb position of multiplier.
2. If current bit is 0 and previous bit is 1, add multiplicand to the accumulator; then shift right, by 1 bit, accumulator and multiplier. The sign bit of accumulator should be similar to the previous status before shift.

3. If current bit is 1 and the previous bit is 0, subtract multiplicand from the accumulator contents. Then, shift right as shown in step 2.

Once all bits of the multiplier are covered, the accumulator and multiplier register together contains the product. Ignore the status of the right end flip-flop used for holding an initial 0 and subsequent lsbs from multiplier.

Example 5.10 Multiply 85×59 using Booth's algorithm.

Multiplicand = $85 = 01010101$; 2's complement = 10101011

Multiplier = $59 = 00111011$

Iteration	Step/Action	Accumulator	Multiplier	Q_{0-1}
0	Initial values	00000000	00111011	0
1	(a) Subtract MD	10101011		
	(Add 2's Comp)	10101011	00111011	0
	(b) Shift right	11010101	10011101	1
2	(a) No action	11010101	10011101	1
	(b) Shift right	11101010	11001110	1
3	(a) Add MD	01010101		
		00111111	11001110	1
	(b) Shift right	00011111	11100111	0
4	(a) Subtract MD	10101011		
		11001010	11100111	0
	(b) Shift right	11100101	01110011	1
5	(a) No action	11100101	01110011	1
	(b) Shift right	11110010	10111001	1
6	(a) No action	11110010	10111001	1
	(b) Shift right	11111001	01011100	1
7	(a) Add MD	01010101		
		01001110	01011100	1
	(b) Shift right	00100111	00101110	0
8	(a) No action	00100111	00101110	0
	(b) Shift right	00010011	10010111	0

product

Ignore

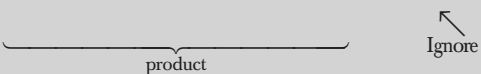
Product = 0001001110010111
= 5015

Example 5.11 Multiply -52×59 using Booth's algorithm.

Multiplicand = $(-52) = 11001100$; 2's complement = 00110100

Multiplier = $59 = 00111011$

Iteration	Step/Action	Accumulator	Multiplier	Q_{0-1}
0	Initial values	00000000	00111011	0
1	(a) Subtract MD	00110100		
		00110100	00111011	0
	(b) Shift right	00011010	00011101	1
2	(a) No Action	00011010	00011101	1
	(b) Shift right	00001101	00001110	1
3	(a) Add MD	11001100		
		11011001	00001110	1
	(b) Shift right	11101100	10000111	0
4	(a) Subtract MD	00110100		
		00100000	10000111	0
	(b) Shift right	00010000	01000011	1
5	(a) No Action	00010000	01000011	1
	(b) Shift right	00001000	00100001	1
6	(a) No Action	00001000	00100001	1
	(b) Shift right	00000100	00010000	1
7	(a) Add MD	11001100		
		11010000	00010000	1
	(b) Shift right	11101000	00001000	0
8	(a) No Action	11101000	00001000	0
	(b) Shift right	11110100	00000100	0



product

Product = 1111010000000100

= -3068

Note: Sign bit of product is 1. Hence the product is in 2's complement.

Many computers use a modified Booth's algorithm and recoding of multiplier as bit-pair recoding which is involved and result in faster multiplication.

5.3.5 Integer Division

The division operation is less frequent but more involved than multiplication. Often we encounter 'divide by zero' situation resulting in an invalid operation. Usually this situation leads to a program exception (interrupt) for taking appropriate action.

The division is more complex arithmetic than multiplication. Given a dividend D and a divisor V , the division of D by V should satisfy the following equation $D = (Q \times V) + R$ where Q is the quotient and R is the remainder.

Easy methods are available for division involving unsigned numbers (D and V). Though there are also some techniques for division with signed numbers, it can be done similar to the division for unsigned numbers and then including sign later. If the operands are in 2's complement form, the usual method followed is first converting them into unsigned numbers before division and then reconverting the result into 2's complement form.

5.3.5.1 Division of Unsigned Integers

Let us first consider the manual method using *paper and pencil* for the example shown in Fig. 5.50 in which $D = 10100011$ (decimal 163) and $V = 1011$ (decimal 11). This method is commonly known as longhand method. The steps are as follows:

$$\begin{array}{r}
 \begin{array}{c} 1011 \end{array} \overline{) \begin{array}{c} 01110 \\ 10100011 \end{array}} \\
 \begin{array}{c} (-) \\ \hline \end{array} \begin{array}{c} 1011 \\ 10010 \\ 1011 \\ 01111 \\ 1011 \\ \hline \end{array} \begin{array}{c} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{array} \\
 \begin{array}{c} 01001 \end{array}
 \end{array}
 \quad
 \begin{array}{l}
 D = 10100011 \\
 V = 1011 \\
 Q = 1110 \\
 R = 1001
 \end{array}$$

Fig. 5.50 Long hand division method

1. Align the divisor below the dividend such that the portion of dividend bits (from msb) covered forms a binary number which is greater than or equal to the divisor.
2. Subtract the divisor from the dividend to form the partial remainder. Put a 1 as quotient bit. (2's complement of divisor is added to the dividend instead of subtraction)
3. Include one more bit (next bit on right side) of the dividend to the right of the partial remainder.
4. Subtract the divisor from the new partial remainder. (2's complement addition is done)

- (a) If remainder is zero or greater than zero (positive number), put 1 as next quotient bit;
 - (b) If remainder is negative (i.e. less than zero), put 0 as next quotient bit; restore the old partial remainder by adding back divisor.
5. Realign the divisor for another subtraction after including one more bit of dividend to the right of the partial remainder. This step is similar to step 1.
 6. Repeat steps 4 and 5, until lsb of the dividend has been included in step 3, and step 4 is completed. Now we have both Q and R .
- $Q = 1110$ (decimal 14); $R = 1001$ (decimal 9).

5.3.5.2 Restoring Division

The manual method is slightly modified in the restoring division method to simplify the data path implementation. Figure 5.51 shows the data path arrangement and Fig. 5.52 displays an algorithm in the flow chart form. Figure 5.53 illustrates the status of registers during the division process. The steps followed for the process are summarized as follows:

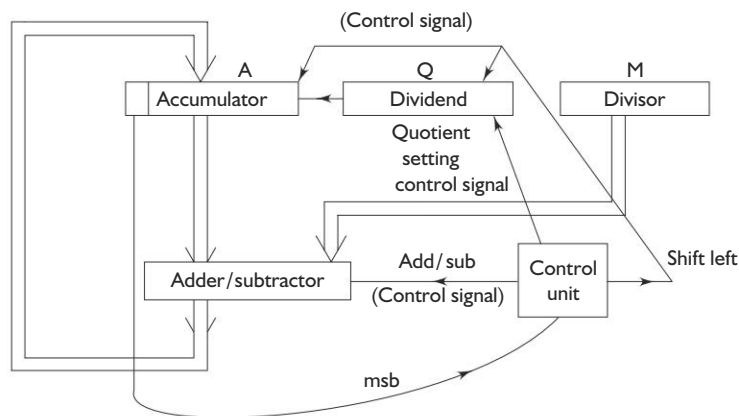


Fig. 5.51 Data path—restoring division

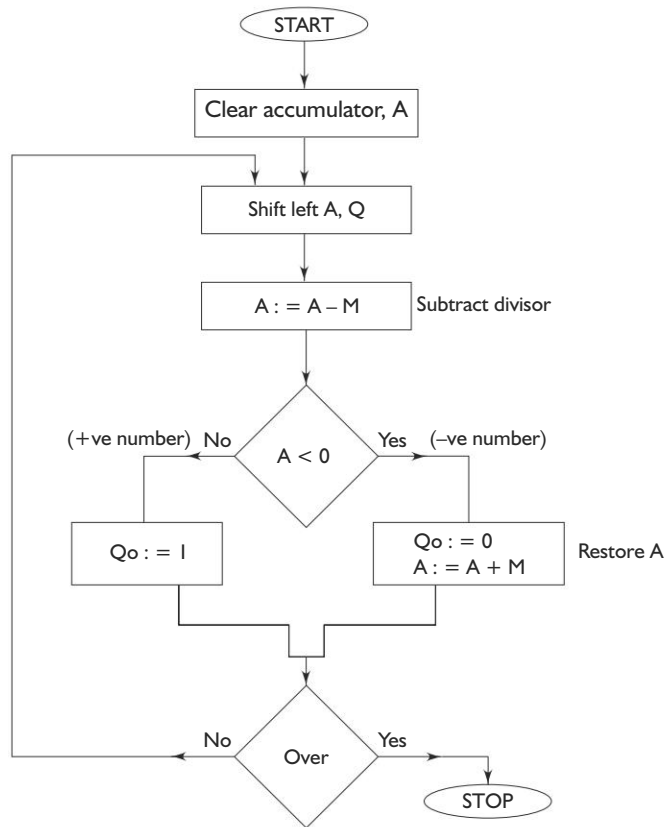


Fig. 5.52 Algorithm for restoring division

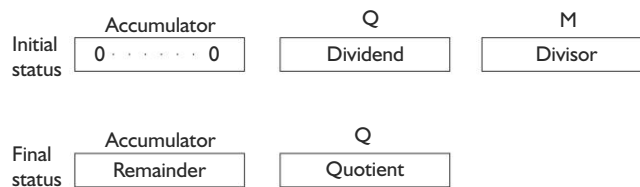


Fig. 5.53 Restoring division registers status

1. Enter the n bit dividend in Q and n -bit divisor in M .
2. Clear A . Contents of A and Q form a single $2n$ bit item.
3. Shift left, by 1 bit, A, Q .

4. Subtract M from A , and replace A by the result. (The divisor should be subtracted from the left half of A)
5. (a) If sign of A is 0, set $Q_0 = 1$ (division is successful)
(b) If sign of A is 1, set $Q_0 = 0$; add M back to current result in A to get back previous value of A (operation unsuccessful; restoring A).
6. Repeat steps 3 to 5 for n times to get remaining bits of Q (1, 2, ... n).
7. The Q contains the quotient and A contains the remainder.

As a standard practice, wherever subtraction is required, addition of two's complement is carried out.

Example 5.12 (Unsigned numbers division)

Divide 163 by 11 using restoring division method.

Dividend, $Q = 163 = 10100011$

Divisor, $M = 11 = 00001011$, $M_{2C} = 11110101$

Iteration	Step/Action	Accumulator (A)	Dividend/Quotient (Q)	Divisor/Remark (M)
0	Initial values	00000000	10100011	00001011
1	Shift left A, Q	00000001	0100011_	
	Subtract A-M	11110101		
	Restore A+M	00001011	01000110	$S_A = 1; Q_0 = 0$
2	Shift left A, Q	00000001	01000110	
	Subtract, M	00000010	1000110_	
	Restore, A+M	00001011	10001100	$S_A = 1; Q_1 = 0$
3	Shift left A, Q	00000010	10001100	
	Subtract M	11110101	0001100_	
	Restore, A+M	00001011	00011000	$S_A = 1; Q_2 = 0$
4	Shift left	00000101	00011000	
	Sub: A-M	11110101		
		00001010	00110000	$S_A = 1; Q_3 = 0$

(Contd.)

Iteration	Step/Action	Accumulator (A)	Dividend/Quotient (Q)	Divisor/Remark (M)
	Restore A+M	00001011		
		00001010	00110000	
5	Shift left Sub: A-M	00010100 11110101	0110000_	
		00001001	01100001	$S_A = 0; Q_4 = 1$
6	Shift left Sub: A-M	00010010 11110101	1100001_	
		00000111	11000011	$S_A = 0; Q_5 = 1$
7	Shift left Sub A-M	00001111 11110101	1000011_	
		00000100	10000111	$S_A = 0; Q_6 = 1$
8	Shift left Sub A-M	00001001 11110101	0000111_	
		11111110	00001110	$S_A = 1; Q_7 = 0$
		11111110	00001110	
	Restore, A+M	00001011		
		00001001	00001110	

Remainder = 9

Quotient = 14

Hence $163 \div 11$ gives, $Q = 14$ and $R = 9$

5.3.5.3 Non-restoring Division

The non-restoring division method is a faster method. It eliminates some unproductive and redundant operations done in restoring division method. The step 5(b) in restoring division method involves restoring back the A. Hence time is wasted in subtraction of M from A in step 4 if the result of subtraction is negative. If we can avoid this wastage by any strategy, we gain speed. This is the basic objective of non-restoring division. Viewed together with step 3, after the step 5(a), we have $R_{i+1} = (2 \times R_i) - M$. On the other hand, after step 5(b), we have $R_{i+1} = R_{i+1} + M$. However, immediately after 5(b), we will be doing steps 3 and 5(a)

for the next cycle. i.e. again we will be doing $R_{i+1} = (2 \times R_i) - M$ for the next cycle. Substituting $R_{i+1} + M$ in place of R_i in this expression, we get $(2 \times R_i) + M$. These are summarized as follows:

If $A \geq 0$, then $2A - M$ (left shift and subtraction)

If $A < 0$, then $2(A+M) - M = 2A+M$ (left shift and addition)

Basically, we are trying to combine current step with next step, thus, eliminating the restore operation for A after an unsuccessful subtraction seen in restoring division method. In case of restoring division, both subtraction and addition are involved, if the quotient is 0. In non-restoring case, either addition or subtraction is done for each quotient bit. Thus, in non-restoring divisions, totally n operations (additions/subtractions) are involved. In restoring division, exact number of additions/subtractions depends on the bit patterns but on an average, $3n/2$ operations are involved assuming that half the times restoring has to be done.

The steps for non-restoring division are as follows:

Step 1(a): If sign of A is 0, shift A and Q by one bit left and subtract M from A ;

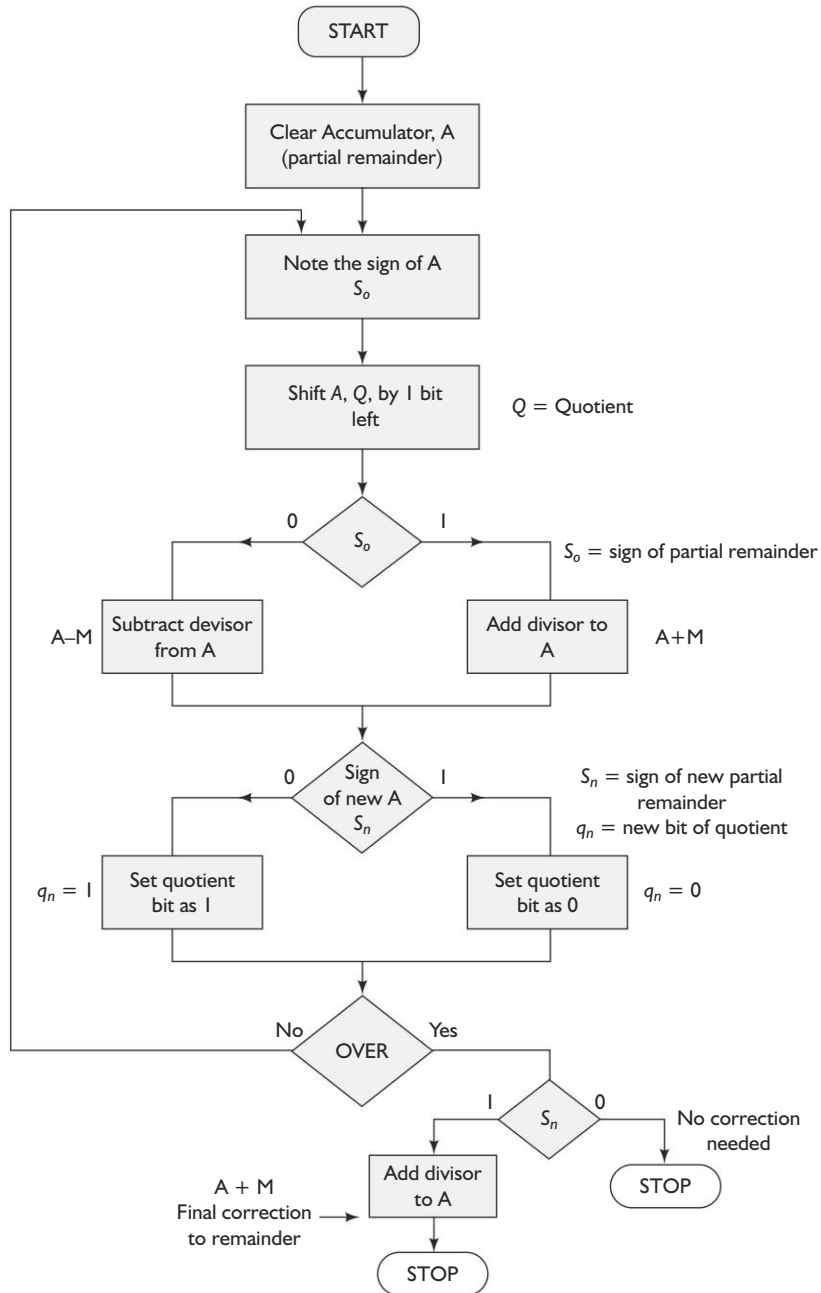
1(b): If sign of A is 1, shift A and Q by one bit left and add M to A .

Step 2: Now, if the sign of new A is 0, set $Q_0 = 1$; otherwise, set $Q_0 = 0$.

Step 3: Repeat the Steps 1–2, n times.

Step 4: If the sign of A is 1, add M to A , to get proper remainder.

For each 0 in the quotient, the restoring division does two operations: subtractions and then addition to restore. But in non-restoring division, for each bit in quotient, only one operation is required: either addition or subtraction. Figure 5.54 gives the algorithm for non-restoring division and fig. 5.55 shows the hardware block diagram. As usual, wherever subtraction is required, addition of two's complement is carried out. Figure 5.56 shows the status of relevant registers before commencing division and after completing division.

**Fig. 5.54** Algorithm for non-restoring division

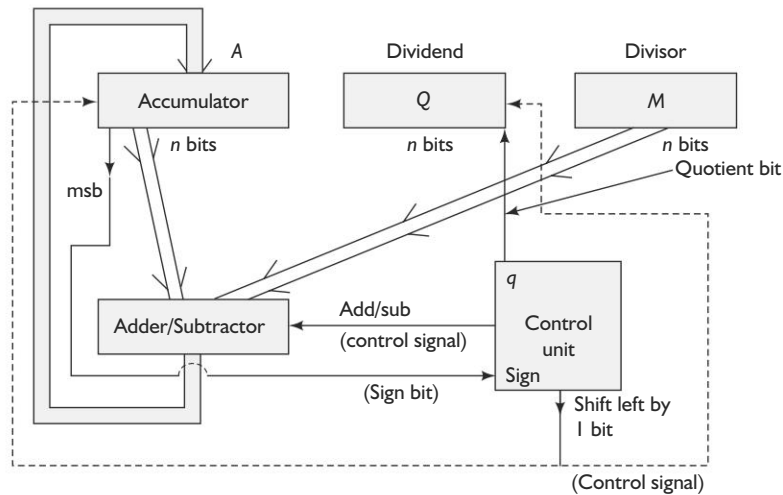


Fig. 5.55 Data path-non-restoring division

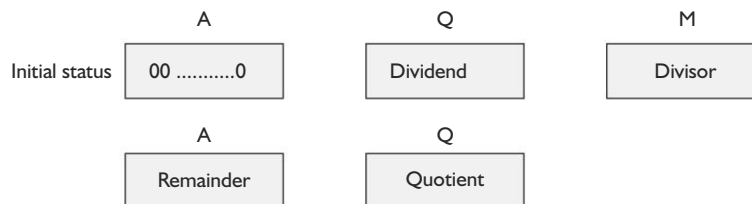


Fig. 5.56 Non-restoring division—registers status

Example 5.13 Divide 163 by 11 using non-restoring division.

Divident = 163 = 10100011; Divisor = 11 = 00001011 2's complement = 11110101

Iteration	Step/Action	Accumulator (A)	Dividend/Quotient (Q)	Divisor(M)/Remark (M)
0	Initial values	00000000	10100011	00001011
1	Shift left A, Q Sub, A-M	00000001 11110101	0100011_	$S_0 = 0$; (A-M)
		11110110	01000110	$S_n = 1$; $q_0 = 0$
2	Shift left A, Q Add A+M	11101100 00001011	1000110_	$S_0 = 1$; (A+M)
		11110111	10001100	$S_n = 1$; $q_1 = 0$
3	Shift left A, Q Add A+M	11101111 00001011	0001100_	$S_0 = 1$; (A+M)
		11111010	00011000	$S_n = 1$; $q_2 = 0$
4	Shift left A, Q Add A+M	11110100 00001011	0011000_	$S_0 = 1$; (A+M)
		11111111	00110000	$S_n = 1$; $q_3 = 0$
5	Shift left A, Q Add A+M	11111110 00001011	0110000_	$S_0 = 1$; (A+M)
		00001001	01100001	$S_n = 0$; $q_4 = 1$
6	Shift left A, Q Sub A-M	00010010 11110101	1100001_	$S_0 = 0$; (A-M)
		00000111	11000011	$S_n = 0$; $q_5 = 1$
7	Shift left A, Q Sub A-M	00001111 11110101	1000011_	$S_0 = 0$; (A-M)
		00000100	10000111	$S_n = 0$; $q_6 = 1$
8	Shift left A, Q Sub. A-M	00001001 11110101	0000111_	$S_0 = 0$; (A-M)
		11111110	00001110	$S_n = 1$; $q_7 = 0$
	Sub. A+M	00001011	00001111	$S_n = 1$, Apply
		00001001	00001110	Correction (A+M)

Remainder = 9

Quotient = 14

5.4 Floating-Point Arithmetic

The floating-point arithmetic has its own rules and needs additional hardware circuits compared to fixed-point datapath. Multiplication and division are easier in floating-point arithmetic than addition and subtraction. The floating-point operands are stored in normalized form which was discussed in Chapter 4. Similarly, the result of an arithmetic operation is stored in normalized form.

Consider two floating-point operands A and B expressed as follows:

$$A = M_a \times r^{E_a} \text{ and } B = M_b \times r^{E_b}$$

where M_a and M_b are mantissas of A and B, respectively, r is the radix (generally 2 or 10) and E_a and E_b are exponents of A and B, respectively. Before performing addition or subtraction between A and B, one of the operands should be reconverted so that both A and B have same exponent value. This is done by shifting the *radix point* (binary or decimal) of that operand.

Example 5.14 Convert the decimal floating point number $1.21300213 \times 10^{15}$ into another format with exponent 18.

$18 - 15 = 3$ Hence the exponent in the new format is higher than present one by 3.

To increase the exponent by three, the mantissa should be shifted right three times.

After the shift, the mantissa becomes 0.00121300. The decimal point has been moved left by 3 positions.

Three digits (2, 1 and 3) from the right extreme are shifted out and lost.

Example 5.15

- What is the value of the number 11010100 in a 8-bit floating-point format defined by Fig. 4.3 in Chapter 4?
- Express decimal number -0.75 in the 8-bit floating-point format indicated in (a).
- What is the decimal equivalent of the following number in IEEE 754 single precision format?

11000001100100000000000000000000

- The 8-bit number is 11010100. As per the format of Fig. 4.3, the number has following parts:
 Sign = 1, hence negative
 Exponent = 101 = 5 in excess 3 format, $5 - 3 = 2$
 Mantissa = 0100

Hence the number is 1.0100×2^2 in standard form (normalised and hidden 1)

$$1.0100 \times 2^2 = 10.100 \times 2^1 = 101 \times 2^0 = (5)_{10}$$

(b) The decimal number is -0.75

-0.75 in decimal becomes -0.11 in binary $-0.11 \times 2^0 = -1.1 \times 2^{-1}$ in normalized form.

Exponent $= -1$; adding the bias $-1 + 3 = +2$.

Representing in 8-bit format of Fig. 4.3

$\overset{\uparrow}{1}$ $\underbrace{010}$ $\underbrace{1000}$
 Sign Exponent Mantissa
 (-ve)

(c) The given number in single precision format is

$\overset{\uparrow}{1}$ $\underbrace{10000011}_{\text{Exponent}}$ $\underbrace{001000000000000000000000}_{\text{Mantissa}}$
 Sign Exponent Mantissa
 (-ve) 131 in excess 127 format

$$\text{Actual exponent} = 131 - 127 = 4$$

Hence the value of the number is

$$-1.0010 \times 2^4 = 10.010 \times 2^3 = 100.10 \times 2^2 = \underbrace{10010.0}_{\text{Exponent}} = (18)_{10}$$

Converting into decimal

$\overset{\uparrow}{1}$ $\underbrace{.0010}$
 $\overset{\uparrow}{1}$ $\underbrace{.125}$

$\therefore$ The value is

$$-1.125 \times 16 = -18$$

5.4.1 Floating-Point Addition

Figure 5.57 show the algorithm for binary floating-point addition.

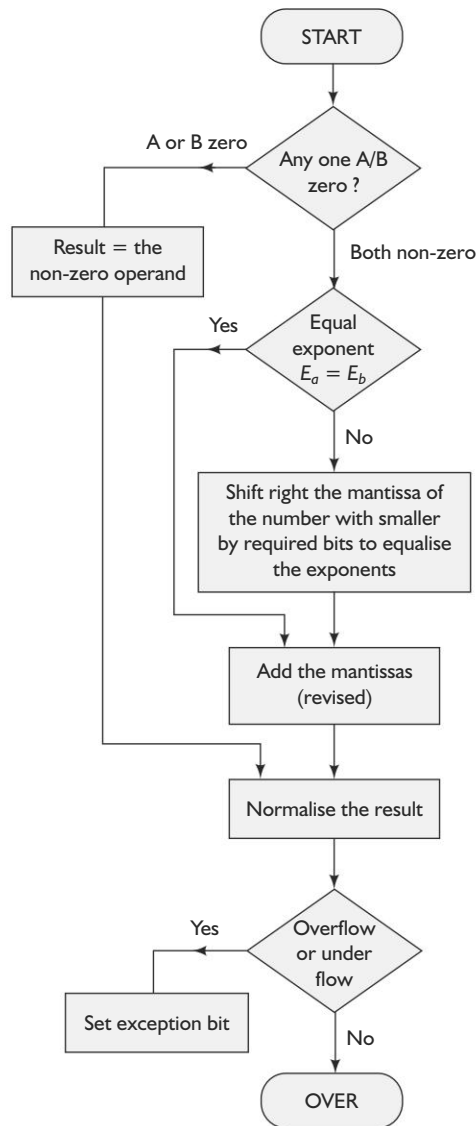


Fig. 5.57 Algorithm for floating-point addition

Consider the addition $(A) + (B)$ where both A and B are floating-point binary numbers. The steps for addition are as follows:

Step1: Check whether A or B is 0; if so, the result is the other operand. Otherwise go to step 2.

Step2: Calculate the difference (ed) between the exponents i.e. $(E_a - E_b)$ or $(E_b - E_a)$.

Step3: Select the number A or B who has smaller exponent.

Step4: Shift right the mantissa of the selected number by number of times equal to the ed i.e. difference between the exponents. Now both numbers have same exponent.

Step5: Perform addition of the new mantissas: $M_a + M_b$.

Step6: The sum is $(M_a + M_b) \times r^{ed}$.

Step7: Normalize the result. This is done by shifting the mantissa bits, so that the binary point is placed to the right of the first non-zero significant (mantissa) bit. This is to ensure that there is no leading zero.

Example 5.16 Add the two decimal floating-point numbers $1.02635126 \times 10^{18}$ and $1.21300213 \times 10^{15}$.

$A = 1.02635126 \times 10^{18}$ and $B = 1.21300213 \times 10^{15}$.

The steps are as follows:

Step 1: Neither A or nor B is 0. So, addition has to be performed.

Step 2: $E_a - E_b = 3$.

Step 3: B has smaller exponent i.e. 15.

Step 4: Mantissa of B has to be shifted right 3 times;

M_b (shifted value) = 0.00121300.

Step 5: $M_a + M_b =$

$$\begin{array}{r} 1.02635126 \\ 0.00121300 \\ \hline 1.02756426 \end{array}$$

Step 6: Result = $1.02756426 \times 10^{18}$; already in normalized form.

Note: In step 3, we can choose the number with bigger exponent also. In that case, in step 4, we have to shift left the mantissa of this number, by three digits, so as to reduce its exponent to 15. However, this will result in less precise result.

Example 5.17 Add 0.75 and 1.8 using binary floating-point representations.

$$A = 0.75 = 1.1 \times 2^{-1} = 0.11 \times 2^0$$

$$B = 1.8 = 1.10011 \times 2^0$$

$$\begin{array}{r} 0.110000 \\ 1.10011 \\ \hline 10.100011 \times 2^0 \\ \underbrace{(2)_{10}} \cdot \underbrace{(.54687)_{10}} \\ = 2.55 \text{ by rounding off} \end{array}$$

5.4.1.1 Guard Digits

When the mantissa is shifted right, some digits at the right most position are lost. To get maximum accuracy of the final result, one or more extra bits known as *guard bits* are included in the intermediate steps. These bits temporarily store the recently shifted out bits from the right most side of the mantissa. When the number has to be finally stored in a register or in a memory as the result, the guard bits are dropped. However, based on the guard bits, the value of the mantissa can be made more precise by the rounding technique discussed in next section.

Example 5.18 Show 0.8 with one guard digit in the 8-bit floating-point format of Fig. 4.3.

$$\begin{aligned}(0.8)_{10} &= 0.11001100 \\ &= 1.1001100 \times 2^{-1} \text{ in normalized form}\end{aligned}$$

In the 8-bit format of Fig. 4.3, only 4-bits are allowed for the mantissa. Including one-bit as guard digit, the number is

00101001 without guard bit

$\begin{array}{ccccccc} 0 & & 010 & & 1001 & \boxed{1} \\ \uparrow & & \underbrace{} & & \underbrace{} & \uparrow \\ \text{Sign} & & \text{exponent in excess 3 format} & & \text{mantissa} & \text{guard bit} \end{array}$

5.4.1.2 Rounding and Truncation

The truncation of a floating point-number involves ignoring (dropping) of the guard bits. This may result in precision loss, the extent of which depends on the value of guard bits. There are many variations of truncation such as chopping, rounding etc. In chopping, the guard bits are simply dropped without any change to other bits. In case of rounding, a 1 is added to the lsb of the mantissa if the msb of the guard bits is 1. After this addition, the mantissa is truncated to n bits. This produces a more accurate result. If the msb of the guard bits is 0, then nothing is added to mantissa and truncation is done.

Example 5.19 Round off the value shown in Example 5.8 without one guard bit.

Number is 00101001 $\boxed{1}$
 $\nwarrow$ guard bit

Since guard bit = 1, add 1 to lsb of the mantissa.

Hence

$$\begin{array}{r} 1001 \\ 1 \\ \hline \end{array}$$

the revised mantissa is 1010

The rounded off number is 00101010

5.4.2 Floating-Point Subtraction

Subtraction of $(A) - (B)$ is similar to addition except that in step 5, subtraction of mantissas is done.

Example 5.20 Subtract $1.21300213 \times 10^{15}$ from $1.02635126 \times 10^{18}$.

Steps 1 to 4 are same as example 4.5

Step 5:

$M_a - M_b =$

$$\begin{array}{r} 1.02635126 \\ 0.00121300 \\ \hline 1.02513826 \end{array}$$

Step 6: Result = $1.02513826 \times 10^{18}$.

This is already in normalised form.

Example 5.21 Perform $0.75 - 0.375$ in binary floating point arithmetic.

$$(0.75)_{10} = 0.11 = 0.11 \times 2^0$$

$$(0.375)_{10} = 0.011 = 0.011 \times 2^0$$

Subtracting the mantissas as,

$$\begin{array}{r} 0.110 \\ 0.011 \\ \hline 0.011 \end{array}$$

The result is $0.011 \times 2^0 = 0.375$

5.4.3 Floating-Point Multiplication

Multiplication of A and B involves the following operation:

$$(M_a \times M_b) \times r^{E_a + E_b}$$

The steps followed are (Fig. 5.58) as follows:

1. Test if A or B is 0; if so the result is 0. Otherwise, go to step 2.
2. Add exponents i.e. $E_a + E_b$.
3. Subtract the bias 127 since both the exponents are in excess 127 format and we have added both.
4. Check whether the exponent overflow or exponent underflow occurs. If so, stop and report by setting up appropriate flags.

5. Multiply the mantissas: $M_a \times M_b$.
6. Normalize the result.
7. Round off the result using the guard digit.

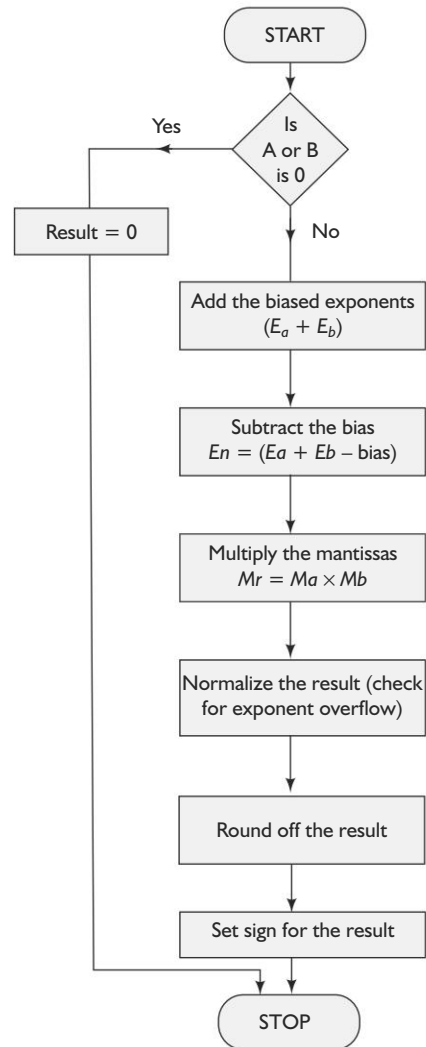


Fig. 5.58 Algorithm for floating-point multiplication

Example 5.22 Multiply the following two numbers shown in 8-bit floating-point format of Fig. 4.3.

$$A = 01010101, B = 10111010$$

Adding the exponents $101 + 011 = 1000$

Subtract the bias $\quad\quad\quad 011$

Result's exponent = $\quad\quad\quad 101$

Multiply the mantissas 1.0101×1.1010

$$= 10.00100010$$

↑
guard bit

Rounding off, 10.01×2^5

Normalizing, 1.001×2^6

Result = $\overset{\uparrow}{1} \quad \quad \quad \underbrace{110}_{\text{Exponent (excess-3)}} \quad \quad \quad \underbrace{0010}_{\text{Mantissa}}$

5.4.3.1 Exponent Overflow

If a positive exponent to be stored is greater than the maximum possible exponent value, it results in exponent overflow. The resulting number is too large to be stored for the given computer. Sometimes, the result may be preserved in denormalized form.

In single precision format, the overflow occurs if a normalized number needs an exponent greater than +127. In the case of double precision, overflow situation occurs if a normalized number requires an exponent greater than +1023.

Example 5.23 Determine the overflow situation for the 8-bit floating point format shown in Fig.4.3.

In this format, we have three bits for the exponent. We use Excess-3 format for the exponent field. Hence exponent field = actual exponent + 3. The following cases are reserved for special values:

1. Value 0 is represented by mantissa zero and exponent (excess-3) is 0
2. Value ∞ is represented by with mantissa 0 and exponent (excess-3) is 7
3. When the exponent (excess-3) is 0 and mantissa is not equal to 0, denormal numbers are represented. A denormal number causes an underflow; it has leading 0's in the mantissa even after maximum possible adjustment in the exponent.

4. When the exponent (excess-3) is 7 and mantissa is non-zero, the value represented is known as a Not a Number (Nan). A Nan is the result of performing an invalid operation like $0/0$ or $\sqrt{-1}$.

The range offered by the exponent field (excess-3) is 1 to 6. Hence, the actual exponent range is -2 to 3 . Hence, overflow occurs if the normalized number's exponent exceeds 3 .

5.4.3.2 Exponent Underflow

If a negative exponent to be stored is less than the minimum possible value, it results in exponent underflow. The resulting number is too small to be stored for the given computer. It may be taken as 0 , approximately.

In a single precision format, the underflow occurs if a normalized number requires an exponent less than -126 . In the case of double precision, underflow situation occurs if a normalized number requires an exponent less than -1022 .

Example 5.24 Determine the underflow situation for the 8-bit floating – point format shown in Fig.4.3.

As seen in example 5.23, the range for the actual exponent is -2 to $+3$. Hence underflow occurs if the normalized number's exponent is less than -2 .

5.4.4 Floating-Point Division

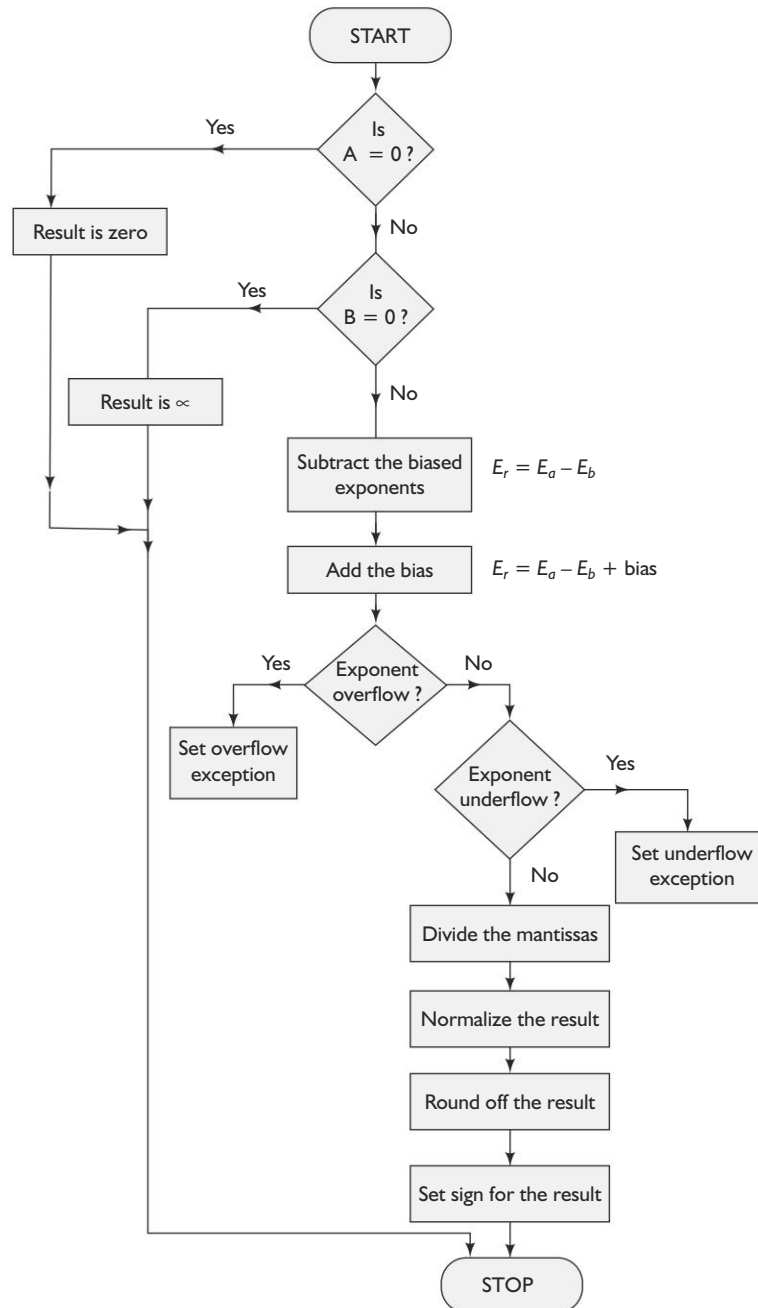
The floating-point division A/B involves the following operation:

$$(M_a/M_b) \times r^{E_a - E_b}$$

The steps are as follows:

1. Test if A is 0 ; if so the result is 0 . Otherwise, go to step 2.
2. Test if B is 0 ; if so the result is ∞ . Otherwise, go to step 3.
3. Subtract exponents: $E_a - E_b$.
4. Add bias 127 since in step 3, both the bias are removed.
5. Check for exponent overflow; if so, stop and report by setting the flag.
6. Check for exponent underflow; if so, stop and report by setting the flag.
7. Divide the mantissas: (M_a/M_b) .
8. Normalize the result.
9. Round off the result.

Figure 5.59 gives the algorithm for floating-point division.

**Fig. 5.59** Algorithm for floating-point division

The datapath for floating-point operations has been already discussed in section 4.31. A typical floating-point ALU has been shown in Fig.4.31.

SUMMARY

The internal circuits of computers are all binary in nature, therefore, binary arithmetic is mostly available in all computers. Fixed-point binary arithmetic is present in all computers. Floating-point format covers a wider range of numbers than the fixed-point numbers. However, the floating-point arithmetic hardware is complex and expensive.

Parallel adder has an adder for each bit. Carry look-ahead adder is faster than ripple carry adder. Subtraction is done by adding 2's complement of the number. The 2's complement addition and subtraction is more convenient in computers than with signed magnitude numbers. Special algorithms used for integer multiplication and division reduces the time for the operation. Decimal arithmetic is desired only if there is extensive data input–output compared to the amount of processing. This is the case with certain business applications. Generally, most of the computers have hardware for binary arithmetic only. Decimal arithmetic is done by conversion of data into binary numbers and reconversion of the result into decimal numbers.

Exceptional situations such as overflow, divide by zero etc. cause program exceptions leading to interrupts and special actions. In these cases, result may not be available or wrong result may be available.

REVIEW QUESTIONS

1. Consider a computer that follows 2's complement arithmetic. The operands have to be stored in 2's complement form. The user presents the data in the decimal form. The data is converted into 2's complement form by whom?
 - (a) Operating system
 - (b) Compiler
 - (c) User program (application)
 - (d) System utilityIs it necessary to have an instruction (in the instruction set) for the decimal to binary conversion?
2. A computer has floating-point arithmetic instructions. But the CPU hardware (datapath) does not have the floating-point ALU. How should the system handle floating-point arithmetic?

3. In some computers, the floating-point ALU is self-contained and has no dependence on integer ALU. What are the advantages and disadvantages of such computers?

EXERCISES

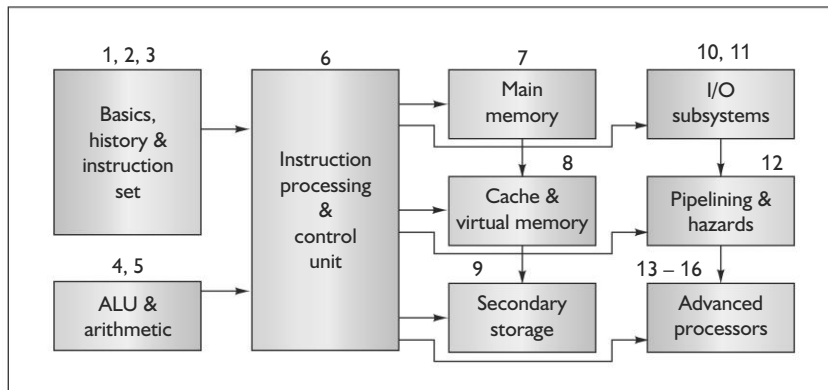
- An 8-bit ALU performed an addition of two operands whose values were 4 and 7. Identify the status of following flags:
 - Carry
 - Zero
 - Overflow
 - Sign
- A memory location contains 1100001011110010. Find the decimal value of this binary word if it represents the following:
 - Unsigned integer
 - 1's complement integer
 - 2's complement integer
 - Signed magnitude integer
- Perform following arithmetic operations in binary using 2's complement representation:
 - $(+43) + (-25)$
 - $(-43) - (-25)$
 - $(+43) + (+25)$
 - $(-43) + (-25)$
- Show the step-by-step multiplication using Booth's algorithm for the following multiplication: $(+12) \times (-9)$.
- Convert following IEEE single-precision floating-point numbers to their decimal values:
 - 00111111111100000000000000000000
 - 01100110100110000000000000000000
- Convert the following decimal numbers to IEEE single-precision floating-point format:
 - 58
 - 3.1823
 - 0.6
 - 38.24
 - 4.02×10^{14}

Restrict the calculation of magnitude values to the most significant 8 bits.
- Perform following division in binary using (a) Restoring division and (b) Non-restoring division:
 $12.35 \div 4.02$
- Determine (a) value of maximum number and (b) value of minimum number for single precision and double precision floating-point format representations.
- Repeat the exercise number 8 for the 8-bit floating-point format discussed in this chapter.

CHAPTER 6



INSTRUCTION PROCESSING AND CONTROL UNIT



Chapter Flow Diagram

OBJECTIVES

- « *Role of control unit*
- « *Basic processing concepts*
- « *Register Transfers*
- « *Single-bus data path and controls*
- « *Instruction cycle and control sequence*
- « *Multiple-bus organization*
- « *Hardwired control unit organization*
- « *Microprogrammed control unit organization*
- « *Nanoprogramming*

KEYWORDS

Microoperations

Control Point

Control Signal

Synchronous Control Unit

Asynchronous Control Unit

Sequencing

Sequence Counter

Firmware

Horizontal Microprogram

Vertical Microprogram

Emit Field

Microdiagnostics

Emulation

Static Decoding

Dynamic Decoding

Nanoprogramming

Hardwired control

Microcode

 6.1 Introduction

The Processor is the most sophisticated functional unit in a computer. Its organization can be compared to a manufacturing firm. The datapath of the processor is equivalent to the 'production floor' or 'shop floor'. The control unit is similar to the 'production planning and scheduling' section. Basically, the datapath has several hardware resources with different capabilities. The control unit is the *brain* or *nerve* centre. It keeps track of the instruction cycle and issues relevant control signals at appropriate time to perform necessary microoperations by the datapath and other units.

The design of the control unit is the main objective of this chapter. Its coverage is closely linked to the contents of chapters 4 and 5. The chapter 4 has covered the design of the datapath. The arithmetic algorithms followed by the control unit are covered in chapter 5. In addition, topics covered in chapter 3 form the basis for this chapter. Hence, the readers may have to refer to these three chapters wherever necessary.

 6.2 Basic Processing Concepts

As discussed in chapter 1, the function of the processor is executing the program stored in the main memory. In simple terms, the processor has to perform instruction cycle continuously except when in 'halt' state. As discussed in chapter 3, the processor has multiple dimensions: System level, Instruction set level, Register Transfer Level and Gate level. Functionally, the processor has two independent units: the datapath and the control unit. The interface between the datapath and control unit is shown in Fig. 6.1.

The control unit is responsible for coordinating the activities inside the computer. Figure 6.2 provides a birds eye view of the overall role of the control unit. The control signals issued by the control unit reach the various hardware logics inside the processor and other external units. The microoperations are performed when the relevant control signals activate the control points. The main memory is controlled by two control signals: memory read and memory write. All I/O controllers receive the control signals IOR and IOW. The datapath is the unit which receives maximum number of control signals. The processor executes a program by performing instruction cycles as shown in Fig. 6.3. Each instruction

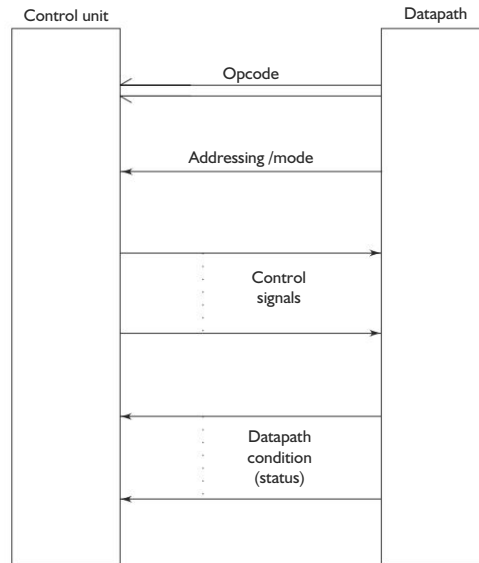


Fig. 6.1 Functional interface between datapath and control unit

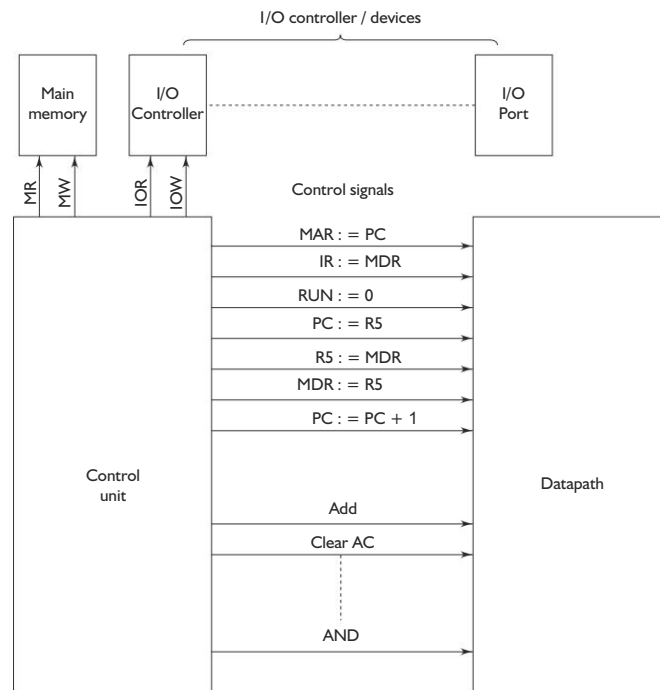


Fig. 6.2 Role of control unit

cycle consists of multiple steps as shown in Fig. 6.4. The overall role of the control unit is summarized in the following points:

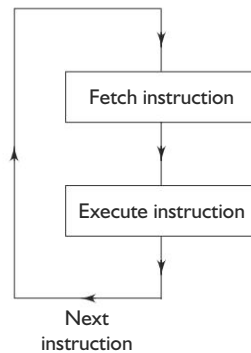


Fig. 6.3 Instruction cycle



IF : Instruction fetch

ID : Instruction decode

OAC : Operand address calculation

OF : Operand fetch

EO : Execute operation

SR : Store result

Fig. 6.4 Steps in instruction cycle

1. Fetch instruction.
2. Decode (analyze) opcode and addressing mode.
3. Generate necessary control signals corresponding to the instruction (opcode and addressing mode) in proper time sequence so that relevant microoperations are done.
4. Go to Step 1 for next instruction.

Apart from regular instruction cycle, the control unit also performs certain special sequences, listed as follows:

1. Reset sequence.
2. Interrupt recognition and branching to interrupt service routine.
3. Abnormal situation handling.

These actions are discussed in detail in the forthcoming sections.

6.2.1 Reset Sequence

The reset signal is generated by the hardware under any of the following situations:

1. *Computer power is switched on:* This activates the Power-On Reset (POR) circuit which generates the power-on reset signal.
2. *Operator presses reset switch/button in the front panel:* This activates the MANUAL RESET signal.
3. *External hardware (outside processor) applies reset signal to processor:* This is a special case to release the processor out of 'HALT' or 'shutdown' situation.

As shown in figure 6.5, for any of the three cases the control unit performs the following four different actions:

1. Resets error flags.
2. Resets Interrupt Enable (IE) flag; the control signal $IE := 0$ is generated for this purpose.
3. Forces a fixed address in the program counter. This address is commonly known as *reset vector*. The control signal $PC := RV$ is generated so that the reset vector address is entered into the program counter. Generally, the reset vector is an 'all zero' address or 'all one' address. Some processors have Built-In Self-Test (BIST) program which is first executed as a confidential measure so that the processor is sure of its reliability. If the BIST is successful, then it enters the reset vector in the program counter. Otherwise, it will *shutdown* itself.
4. Sets RUN/HALT flip-flop. The control signal $RUN := 1$ does this.

The net effect of the actions taken in steps 3 and 4 initiate the processor to do the instruction processing starting from the reset vector. Once the reset sequence is completed, the hardware is under the control of the software.

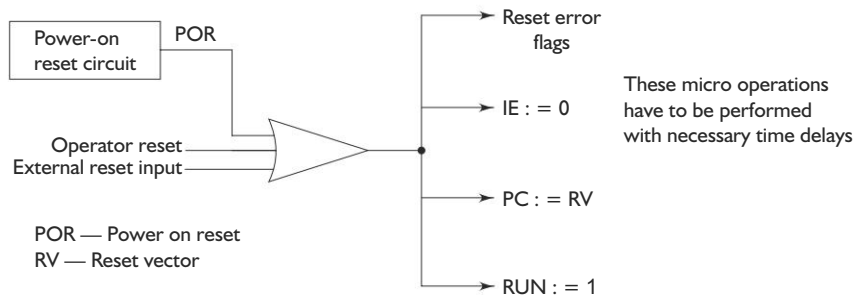


Fig. 6.5 Reset action

6.2.2 Interrupt Recognition and Servicing

Interrupt can occur at any time. It is a request to the processor asking for its attention to perform some special action by branching to the Interrupt Service Routine (ISR). Normally, the control unit checks for the presence of interrupt request before fetching a new instruction after the completion of previous one. In case, the interrupt enable flag is 0, the control unit skips this step and the interrupt remains pending. On sensing an interrupt request, the control switches to ISR. Figure 6.6 shows the modified instruction cycle and interrupt servicing by the control unit.

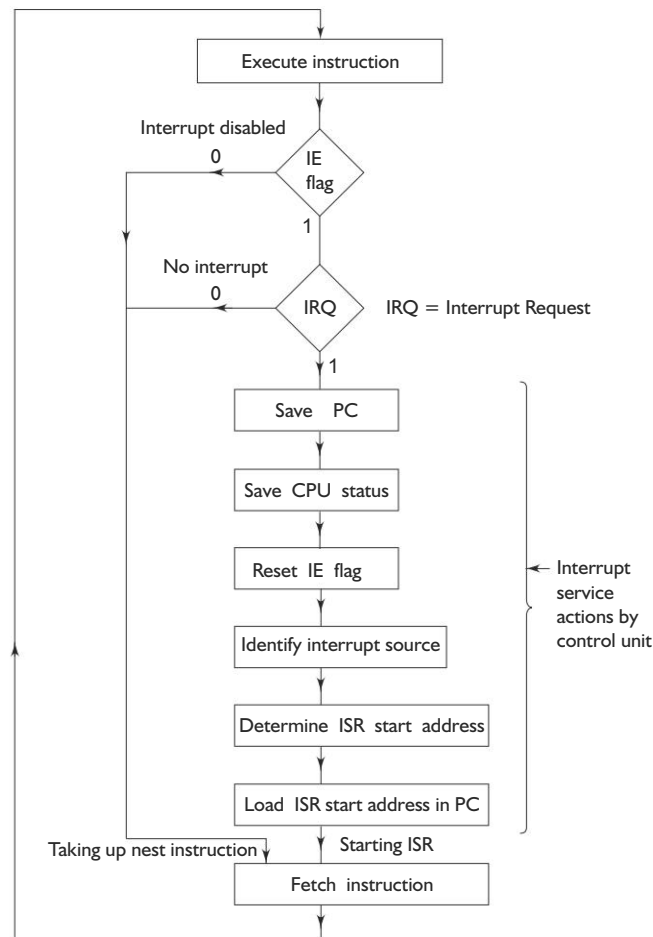


Fig. 6.6 Instruction cycle and interrupt servicing

6.2.3 Abnormal Situation Handling

At times, the control unit faces some serious abnormal situation due to which, further processing is totally unreliable. Two such cases are *double error* and *machine check* discussed below:

Double error: The processor has sensed an exceptional condition such as overflow, illegal opcode etc., and while taking action against this (interrupt servicing), another exception occurred resulting in a ‘double error’ situation. In such a case, the control unit performs ‘shutdown’ by resetting RUN/HALT flip-flop.

Machine check: While the processor is performing a read/write bus cycle, the error detection hardware has noticed an error due to which the current bus cycle is unsuccessful, the control unit recognizes this as a ‘machine check’ condition. It performs a diagnostic scan by saving the CPU status in the special machine check registers and then branches to machine check service routine.



6.3 Instruction Cycle and Processor Actions

Since the instruction set of a CPU may support a variety of addressing modes and operand formats, the control unit is designed for taking care of all possibilities. Some of the issues and decisions taken by the control unit during the instruction cycle are discussed here:

6.3.1 Instruction Fetch

This phase involves fetching the instruction from program memory from the location whose address is available in program counter. The instruction read from memory is loaded into the instruction register, IR. This action is symbolically represented as $\langle PC \rangle \rightarrow IR$. The same microoperation can also be represented as $IR \leftarrow \langle PC \rangle$.

If the instruction length is more than the memory word length, the control unit performs more than one memory read cycle to fetch the instruction.

Example: Instruction length is 4 bytes and memory word length is 2 bytes, two words have to be fetched.

If the width of the bus between the CPU and memory is less than memory word length, the control unit performs more than one bus cycle to get one memory word.

Example: The data bus is 16 bits and memory word length is 32 bits, two bus cycles are needed to fetch one word.

How does the control unit know about the length of the instruction being fetched? The instruction length is indicated in the opcode of the instruction. Hence, the control unit knows about the instruction length only after fetching the first word of the instruction. Based on the opcode, the control unit decides about the number of words to be fetched from the memory. Figure 6.7 shows the logic followed by the control unit during instruction fetch. If the instruction length is 4 bytes, then the PC should be incremented by 4. This is represented either as $\langle PC \rangle + 4 \rightarrow PC$ or as $PC \leftarrow \langle PC \rangle + 4$. The datapath relevant to the instruction fetch sequence along with the control signals is shown in Fig. 6.8. This figure has been already discussed in section 4.14. We have also seen the control signals and datapaths for certain simple instructions in section 4.14.

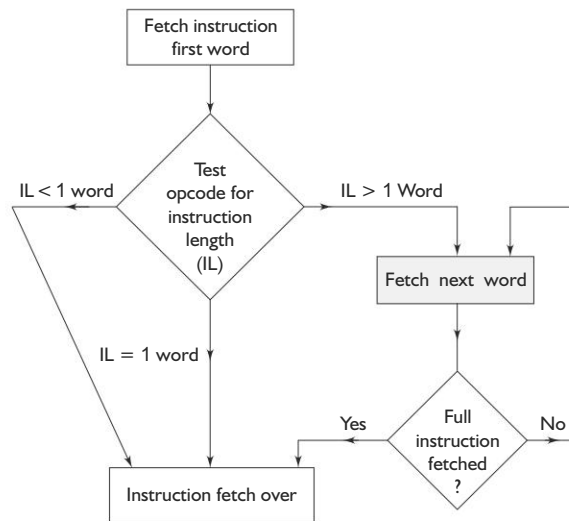
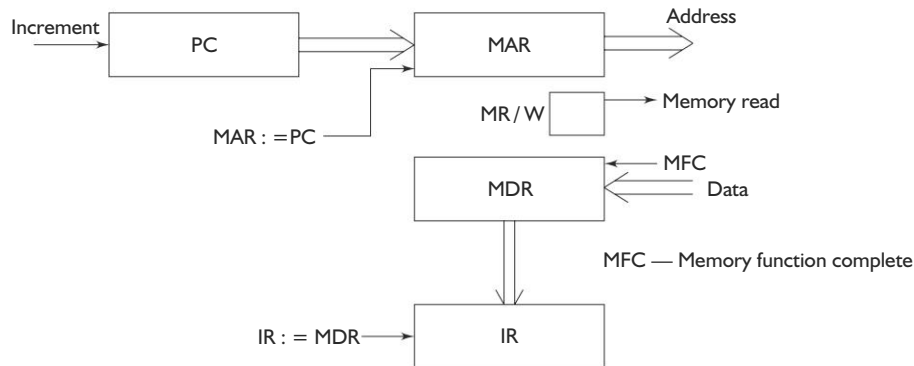


Fig. 6.7 Instruction fetch—multiple words

6.3.2 Operand Address Calculation

The control unit understands the exact location of the operand either from the opcode or from the mode field in the instruction. There are three common locations for the operand:

1. Instruction itself (as immediate operand)
2. Processor register
3. Main memory

**Fig. 6.8** Datapath for instruction fetch

The operand address is indicated in the instruction in one of the addressing modes (as seen in Chapter 3) such as direct addressing, indirect addressing, relative addressing, index addressing, etc. Depending on the addressing mode, the procedure for the operand address calculation differs, as discussed in Chapter 3. The control unit accordingly takes appropriate steps, for calculating the operand address.

6.3.3 Operand Fetch

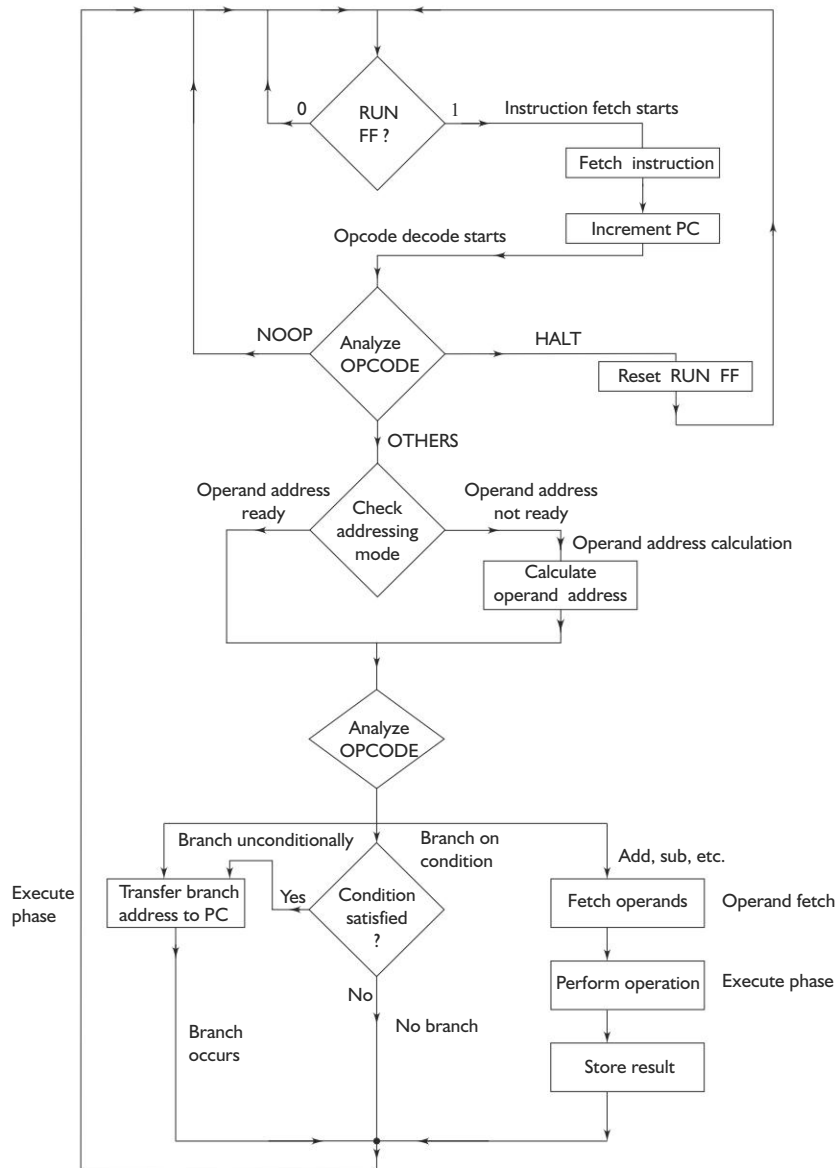
If the operand is in main memory, the CPU performs memory read cycles. The number of main memory read cycles depends on two factors: operand length and memory word length, similar to instruction fetch sequence. The control unit should keep track of the memory cycles. If the operand is in a processor register, it is read quickly compared to an operand from the main memory.

6.3.4 Execute Operation

The control unit is designed to activate various macrooperations as defined by its instruction set. From the opcode, the control unit understands the exact action (microoperations) sequence to be performed during this stage. This varies from no microoperation to multiple microoperations. For some instructions, the microoperations are the internal actions occurring in the processor. For others, the microoperations are performed in external units such as memory, I/O controllers etc. Table 6.1 defines actions required in the execution stage for some of the instructions.

TABLE 6.1 Instructions and actions required in execute stage

S. No.	Opcode	Actions in execute stage
1	NOOP	Nil
2	HALT	Reset RUN flip-flop
3	EI	Set IE flip-flop
4	DI	Reset IE flip-flop
5	INC	Increment accumulator contents
6	CMA	Complement accumulator contents
7	LDA	1. Read from the memory address given in the instruction 2. Enter the memory data into accumulator
8	STA	Write accumulator contents in memory address given in the instruction
9	SKIP	Increment program counter
10	BUN or JUMP	Enter the 'branch address' (given in the instruction) in program counter
11	BSA (branch and save return address)	1. Store PC contents in the address indicated in the instruction 2. Enter the address in PC 3. Increment PC
12	ISZ (increment and skip if zero)	1. Read from the memory address given in the instruction 2. Increment the memory data (in 2's complement form) by 1 3. Store the result in the memory address 4. If the result is zero, increment PC
13	IN	Perform one I/O read bus cycle; input operation
14	OUT	Perform one I/O write bus cycle; output operation
15	INT	1. Store PC and CPU flags in stack 2. Read the interrupt vector 3. Read from the memory address given by the interrupt vector 4. Enter the memory data in PC
16	IRET	Read from the stack and store in PC and CPU flags
17	CIR (circulate right)	Shift accumulator contents by right. The E flip-flop should be copied to msb
18	CIL (circulate left)	Shift accumulator contents by left. The E flip-flop should be copied to the lsb
19	AND	Perform bitwise AND of operands
20	ADD	Perform addition of operands
21	EXECUTE	1. Read from the memory address given in the instruction 2. Enter the data in IR 3. Extend the execute phase
22	JZ	If the accumulator is zero, enter the branch address in PC

**Fig. 6.9** Overall decision making in control unit

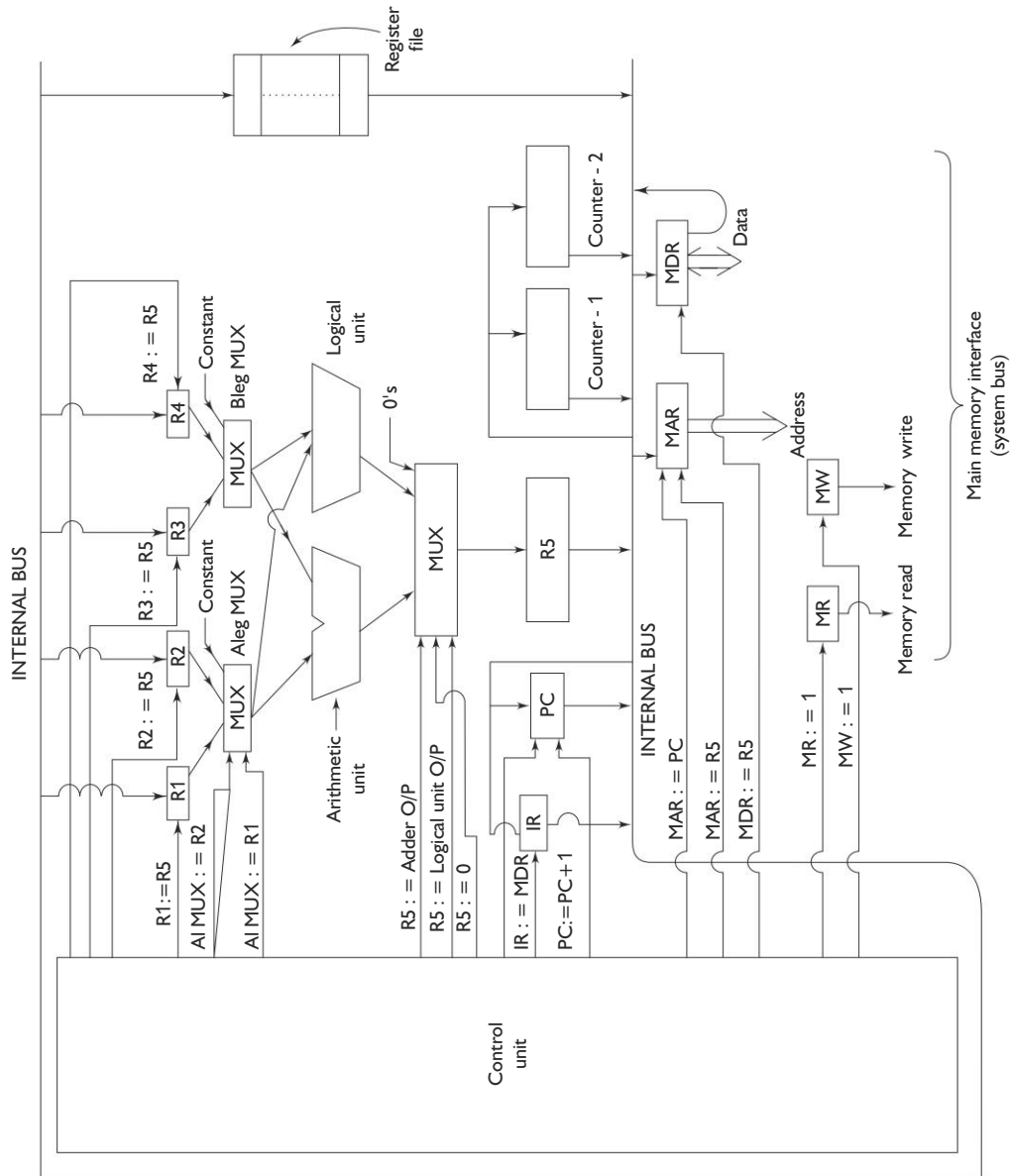


Fig. 6.10 Partial set of control signals

Figure 6.9 shows general logics followed by the control unit for some of the instructions while, Fig. 6.10 shows some of the control points and the relevant datapath in a typical mainframe CPU whose organization has been discussed in section 4.11

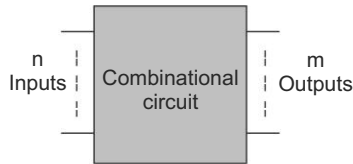
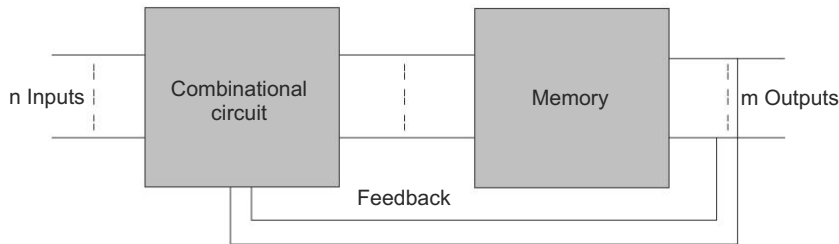
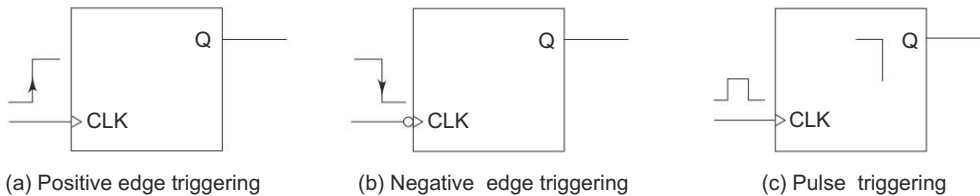
In minicomputers and microcomputers, the internal bus interconnects the various hardware units inside the processor. In section 4.10, we have discussed the datapath of a minicomputer. The reader can try to create a diagram similar to Fig. 6.10 for the figure 4.22 showing the control points and relevant control signals for the minicomputer. We have discussed three different types of datapaths of internal bus based processors in section 4.5. The control points of these are discussed in the next two sections. Irrespective of the type of datapath organization, the control signals should be generated at the correct time: neither earlier nor later. This timing is taken care by clock and synchronization.

6.3.5 Clocking and Synchronization

As mentioned in section 1.6.4, the clock signal serves as a timing reference to all sections of the computer. The clock signal controls and synchronizes the operation of individual circuits so as to fulfill the intended purpose of various circuits.

The digital circuits are classified into two types, based on their behaviour and timing: combinational circuits (Fig. 6.11) and sequential circuits (Fig. 6.12). The output of a combinational circuit at any instant depends only on the inputs present at that instant and has no relevance to previous inputs or previous output. On the other hand, the output of a sequential circuit at any instant depends on both the inputs at that instant and on the previous inputs or previous output. Hence, a sequential circuit can be viewed as a combinational circuit with memory. Multiplexor, demultiplexor, decoder and encoder are some examples of combinational circuits. Flip-flops, counters and shift registers are some examples of sequential circuits.

A digital circuit *wakes-up* to the clock input and immediately recognizes the data inputs so as to generate the output according to the function of the digital circuit. The clock is a chain of uniform pulses occurring at uniform intervals. Hence what a digital circuit receives at its clock input is a clock pulse of certain duration. A given digital circuit can be 'woken-up' (gets triggered) either by any one of the edges of the clock pulse or by the complete pulse. Accordingly the digital circuits are classified as 'edge triggered' or 'pulse triggered' (also known as level triggered). A given edge triggered circuit is designed as positive edge triggered or negative edge triggered so that it is triggered only at one of the edges of the clock. Fig. 6.13 shows the symbols for the three types of triggering used in flip-flops.

**Fig. 6.11** Combinational circuit**Fig. 6.12** Sequential circuit**Fig. 6.13** Types of triggering

6.3.6 Single-cycle and Multi-cycle Design

Several microoperations are performed during any instruction cycle. These have to be done in proper sequence as a series of steps. The control unit is designed to generate and issue the control signals in the desired sequence. The number of steps and the exact sequence (order) depends on the organization of the datapath. In some datapaths, entire sequence (all steps) of microoperations for any instruction can be performed in a single clock cycle. In some datapaths, each step is performed in one clock cycle. In other words successive clock cycles are used to perform microoperations of successive steps.

In the case of single-cycle design, the clock pulsewidth should be long enough to take care of the most complex instruction. Even though many instructions are completed in shorter times, the control unit should wait till the entire clock pulse duration is complete. Hence the single cycle design is inefficient.

In multicycle design, each step in the instruction takes one clock cycle. Hence multiple clock cycles are needed for each instruction. Different instructions need different number of clock cycles. Also a given hardware unit (register, counter etc.) can be used more than once during an instruction execution provided it is used on different clock cycles. In other words, a hardware unit is shared for more than one use in different clock cycles. This reduces the hardware cost. The multicycle design is also known as multiphase clocking.



6.4 Single-bus Processor

A broad organization showing the internal and external buses of a simple microprocessor was presented in Fig.1.44 in chapter 1. Depending on the capability level of the processors, the extent of hardware resources inside the processors and the nature of internal bus linking the hardware units vary. A typical organization of the datapath of a single-bus based processor is shown in Fig. 6.14. Having seen the datapath of a minicomputer (Fig.4.22) and that of a mainframe (Fig.4.23) in chapter 4, the readers will find this figure self explanatory. The salient features of this datapath are listed below:

1. The control unit decodes the instruction available in the Instruction register, IR and generates the various control signals. The control signals are not shown in this figure.
2. The program counter, PC, has bi-directional path with the internal bus. During instruction fetch, the contents of PC enters the internal bus so that it can ultimately reach the memory address register, MAR. During the execution of certain instructions such as branch instruction, the branch address specified by the instruction is entered into the PC via the internal bus. During each instruction fetch, the contents of PC are incremented so as to point to the address of the next successive instruction. This incrementation facility can be built-into the PC or alternatively the ALU can be used for this purpose. However, the later approach leads to a very slow execution.
3. At the end of instruction fetch, the contents of external data bus consist of the instruction supplied by the memory. It is loaded into the MDR. From the MDR, the instruction reaches the IR via the internal bus.

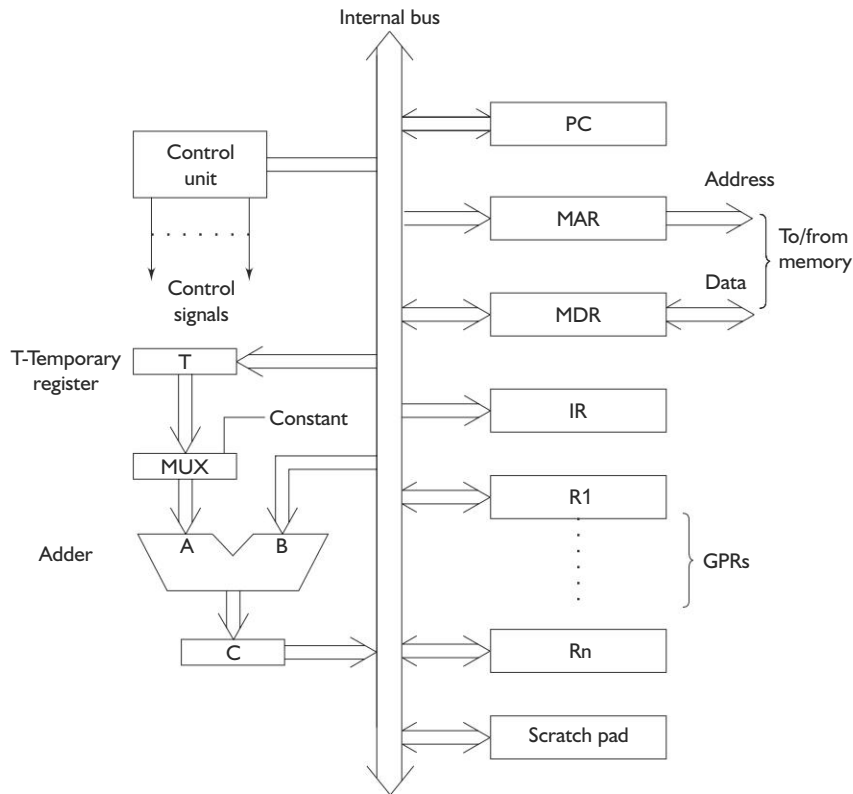


Fig. 6.14 Single bus based datapath

4. The GPRs, R1 to Rn, are program addressable general purpose registers used for multiple purposes: storing operands, addresses, constants, results etc.
5. The scratch pad is collection of internal working registers of the processor not accessible to the programmer. Only the control unit can make use of them for temporary storage of operands, intermediate results or constants during an instruction cycle.
6. The ALU has two inputs: A and B. The B input is the contents of internal bus whereas the A input is the output of a 2-to-1 multiplexer. The multiplexer has two data inputs: the output of the temporary register, T, and a constant. The output of the adder is entered into the output register, C. The registers T and C are not available to the programmer.

7. For any memory access by the processor, the MAR and MDR act as the interface with the memory. The input to the MAR is the contents of the internal bus. Depending on the type of memory access, it could be an instruction address or an operand address. The output of the MAR is sent to memory through the external address bus. The MDR is bi-directional both with the internal bus and with the external bus. During memory read operation, the contents of external data bus enter into the MDR. During memory write operation, the contents of the MDR enter the external data bus.

The movement of information through the internal bus is coordinated by the control signals issued by the control unit. The exact control signal(s) made active at a given time corresponds to the operation code of the instruction being executed. As pointed out in chapter 1, the processor has to perform several microoperations to execute a single instruction. Some common types of microoperations are listed below:

1. Clearing a register
2. Incrementing a counter
3. Adding two registers
4. Transferring information from one register to another register
5. Complementing a register
6. Shifting information in a register
7. Reading from memory
8. Writing into memory

6.4.1 Register Transfers

As discussed in Chapter 3, the microoperations are classified into four major types:

1. Register transfer microoperations
2. Arithmetic microoperations
3. Logic microoperations
4. Shift microoperations

As pointed out in Chapter 4, during the instruction cycle, the control unit issues a sequence of control signals with different time delays. It is the clock signal that helps the control unit to generate the control signals at appropriate instants of time. When a control signal activates a control point, the corresponding microoperation is executed by the datapath.

Fig.6.15 shows some control points for the single-bus processor seen in fig.6.14. Consider the control points for R1. The R1in is control signal that controls input to R1. When R1in is active, the contents of internal bus enter into R1. Whenever, R1in is inactive, the contents of R1 remain unaffected. Similarly, transfer of data from R1 to the internal bus takes place when R1out is active. Whenever, R1out is inactive, the R1 contents do not affect the internal bus. Consider data transfer from R1 to R3 as indicated by the block diagram in Fig.6.16. To achieve this, following two control signals should be generated:

1. First, R1out should be made active. As a result, the contents of R1 enter the internal bus.
2. Next, the R3in should be made active. Immediately, the contents of internal bus enter R3.

By making use of the clock signal, the control unit takes care of activating the R1out first and activating the R3in next. Being registers, R1 or R3 consist of multiple flip-flops with common clock control as seen in Fig.5.13. Each bit is implemented by a D flip-flop similar to that seen in Fig.5.10. The actual circuit diagram for each bit is shown in Fig.6.17. The D flip-flop is a positive edge triggered flip-flop. At the instant of positive edge (rising edge) of the clock, the flip-flop is triggered and the D input reaches the Q output. In other words, the flip-flop is set if D input is 1 (HIGH) and reset if the D input is 0 (LOW).

The 2-to-1 multiplexer selects one of the data inputs (either the A or B input) depending on the SEL control input and connects the selected input to the output (Y) of the MUX. If the SEL is 0, the MUX selects the A input. Otherwise, the MUX selects the B input. Hence when R3in is 0 (LOW), the flip-flop output reaches the D input and on getting triggered by the rising edge of the clock, the Q takes the state of D. In other words, the flip-flop output remains unchanged. When the R3in is 1 (HIGH), the MUX selects the B input and connects to the output. Since the B input of the MUX is D2 bit of the bus, whatever is the status of this bit on the bus, it is transferred to the Q via the MUX.

The Q output of the flip-flop is also given as input to a tri-state buffer. When R3out is 1 (HIGH), the input of this gate is transferred to the output; in other words, this bit of the bus becomes 0 or 1 depending on whether the flip-flop's output is 0 or 1. When the R3out is 0, the tri-state buffer isolates the output from the input. In addition, its output takes high impedance state due to which it is electrically disconnected from the bus though the physical connection still exists. It can be observed that each bit is really a bus since it is common to several sources and destinations. Hence when we use the term internal bus, we imply a group of buses. Thus, the bus shown in Fig.6.17 is actually a collection of 8 buses each of which is for one bit.

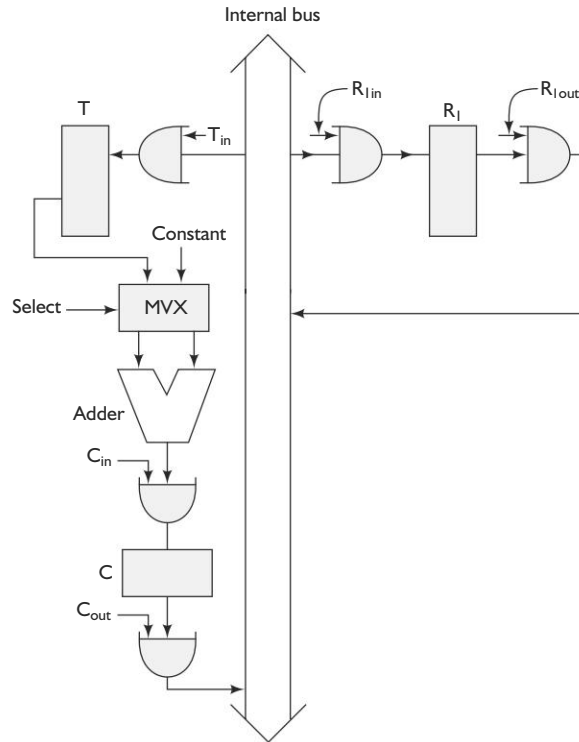


Fig. 6.15 Control points and gating in single-bus datapath

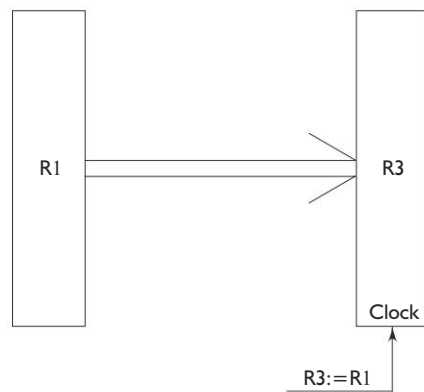


Fig. 6.16 Block diagram for register transfer

TABLE 6.2 Microoperations for ADD R1,R2

Step No.	Control signal	Microoperation	Remarks
1	R1out	The contents of R1 enter the internal bus	The first operand is being fetched
	Tin	The contents of the internal bus enter the T register	The first operand reaches one input of the MUX
2	R2out	The contents of R2 enter the internal bus	The second operand reaches the B input of the adder
	Select T	The select control of the MUX enables selection of T register (and not the constant)	The first operand reaches the A input of the adder
	Add	The adder performs addition of the two operands	After the adder delay, the result is available at the output of the adder
	Cin	The adder output enter the C register	The result reaches the adder output register, C
3	Cout	The contents of C enter the internal bus	The result is on the bus
4	R1in	The contents of the bus enter R1	The result is stored in the destination register

6.4.3 Control Sequence for Processor-Memory Transfers

We have seen the basics of CPU-Memory interface in chapter 1 and seen, in chapter 4, the register transfer operations in the processor for memory read and memory write operations. Basically, the processor initiates the memory read/write operation by first issuing the address of the memory location. The control signal, memory read (Fig.6.18) or memory write (Fig.6.19), is generated by the processor to indicate the type of operation initiated by the processor. In the case of *synchronous memory interface* (Fig.6.20), it is the processor's responsibility to finalise when to wind up the read/write operation with the knowledge about the speed of the memory. In the case of asynchronous memory interface (Fig. 6.21), the memory logic has the responsibility of generating the signal MFC to indicate that the memory operation is over and the processor can wind up the operation. The control points for the MDR is shown in Fig.6.22.

Assuming an *asynchronous memory interface*, let us consider the control sequence for memory read operation for the instruction LOAD R1, <R3>. The contents of memory address in R3 should be read and loaded into R1. The microoperation steps are as follows:

1. $MAR \leftarrow \langle R3 \rangle$
2. Start 'memory read' operation
3. Wait for 'MFC' signal from memory
4. Enter external databus contents into MDR
5. Transfer MDR contents into R1

Being asynchronous memory read operation, the number of clock cycles needed is not known to the processor and it depends on the speed (access time) of the memory. (In the case of synchronous memory read operation, the processor waits for a specific number of clocks matching with the memory access time). The control sequence for the above microoperation steps is given below:

1. R3out, MARin, MR
2. Loop MFC
3. MDRin,E
4. MDRout, R1in

The step2 indicates that the processor keeps waiting for the signal MFC from the memory. The MDRin,E indicates that the transfer is from the external data bus to the MDR. A typical timing diagram for memory read operation is shown in Fig.6.23.

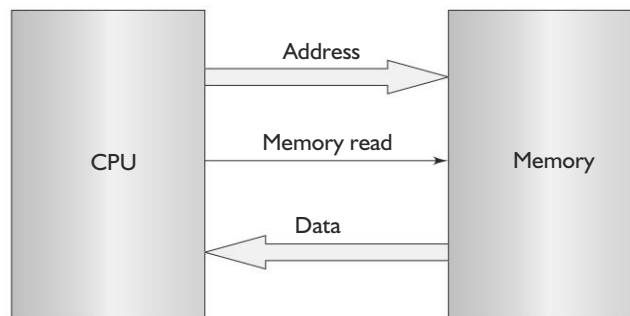


Fig. 6.18 Memory read operation

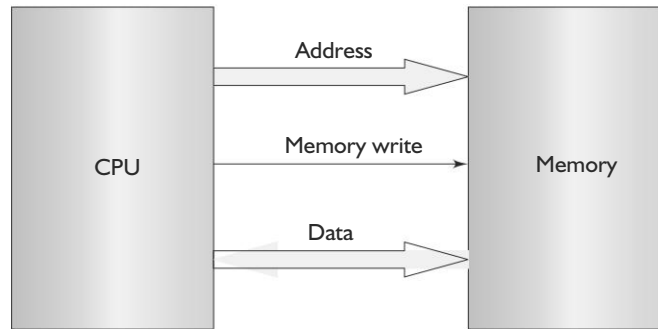


Fig. 6.19 Memory write operation

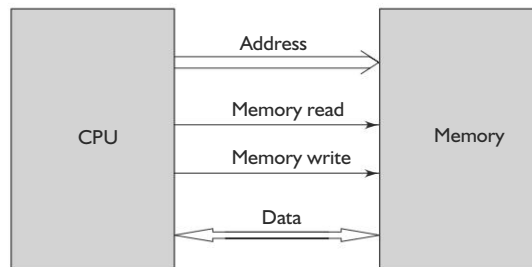


Fig. 6.20 Synchronous memory interface

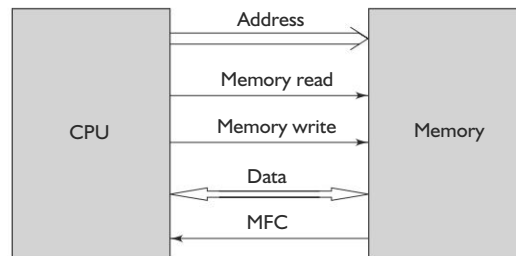


Fig. 6.21 Asynchronous memory interface

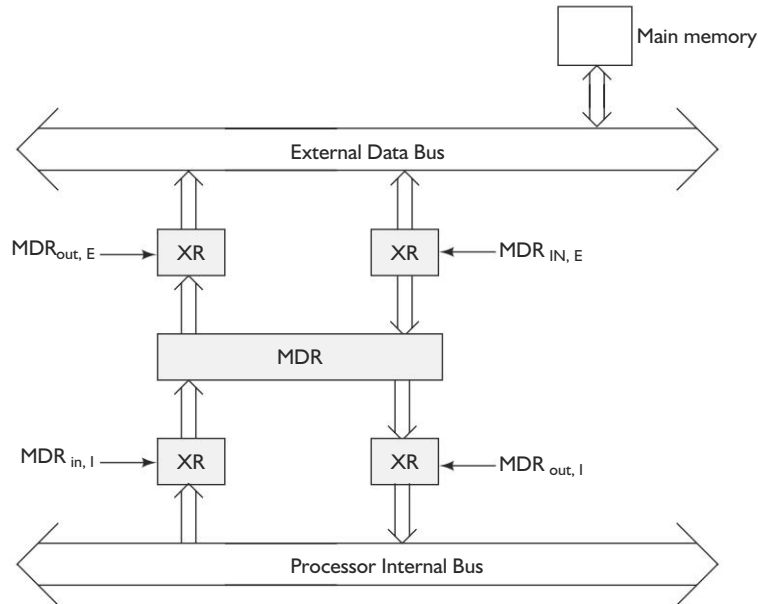
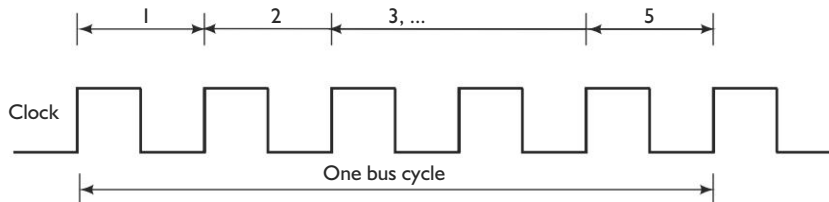


Fig. 6.22 Control points for MDR



- Clock 1: Memory address put on external address bus
- Clock 2: 'Memory read' signal put on external control bus
- Clock 3: Number of clocks depend on memory access time onwards
- Clock 4: Two clock cycles are assumed here
- Clock 5: Winding up: All signals can be removed from the external bus

Fig. 6.23 Timing diagram for memory read operation

6.4.3.1 Memory Write Sequence

The memory write sequence differs slightly from the memory read sequence. Consider the instruction STORE R1, <R3>. The contents of R1 should be stored in the memory address given by R1. The microoperation steps are as follows:

1. $MAR \leftarrow \langle R3 \rangle$
2. Put R1 contents on the internal bus
3. Transfer the contents of internal bus into MDR
4. Start memory write operation
5. Wait for MFC signal from the memory

The control sequence for the above microoperation steps is given below:

1. R3out, MARin
2. R1out, MDRin, MW
3. MDRout,E
4. Loop MFC
5. End

The control MDRout,E in step 3 indicates that the MDR contents should be put on the external data bus. Being asynchronous memory operations, the number of clocks for which the processor remains in control step 4 is not fixed and depends on the memory access time. In the case of synchronous memory interface, the processor remains in control step 4 matching with the access time of the memory. Fig. 6.24 shows a typical timing diagram for write operation on asynchronous memory interface.

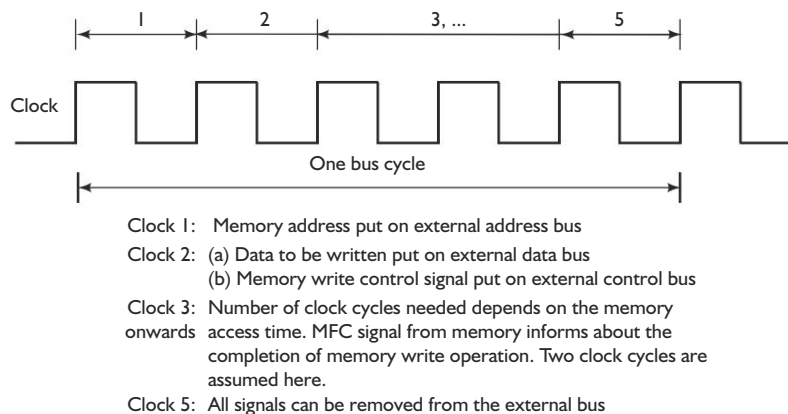


Fig. 6.24 Timing diagram for memory write operation

6.4.4 Control Sequence for Complete Instruction Cycle

Having seen the control sequence for certain simple operations, we are now better prepared to consider an entire instruction cycle consisting of both fetch and execute sequences. An instruction cycle involves instruction fetch, instruction decode, operands fetch, performing the operation and storing the result. Consider the instruction `ADD <R1>, R2`. The microoperation sequence for single-bus processor for this instruction is given below:

1. $MAR \leftarrow PC$
2. MR
3. Loop MFC
4. $IR \leftarrow MDR$
5. $MAR \leftarrow R1$
6. MR
7. Loop MFC
8. $T \leftarrow MDR$
9. $Bus \leftarrow R2$
10. Add
11. $C \leftarrow \text{Adder O/P}$
12. $R2 \leftarrow C$

The control sequence for the above microoperation sequence is given below:

1. PCout, MARin, MR
2. Inc PC
3. Loop MFC
4. MDROUT, IRin
5. R1out, MARin, MR
6. Loop MFC
7. MDROUT, Tin
8. Add
9. Cin
10. Cout, R2in, End

6.4.5 Control Sequence for Branch Instructions:

A simple branch instruction gives the branch address as part of the instruction itself. In this case, the microoperation required in execute phase is transferring the branch address into PC replacing the contents got by incrementing the PC at the end of fetch phase. As pointed out in chapter 4, there is only one microoperation for such a simple branch instruction. In long version of branch instruction, the instruction gives only the offset address. The branch address is calculated as a sum of the updated contents of the PC and the offset address in the instruction. The sequence of microoperations is as follows:

1. $MAR \leftarrow PC$
2. MR
3. Loop MFC
4. $IR \leftarrow MDR$
5. $T \leftarrow JA (IR)$
6. $Bus \leftarrow PC$
7. Add
8. $C \leftarrow Adder\ O/P$
9. $PC \leftarrow C$

The above microoperation sequence can be achieved by the following control sequence:

1. PCout, MARin, MR
2. Inc PC
3. Loop MFC
4. MDRout, IRin
5. Tin, IR (JA) out, SEL MUX
6. PCout, Add, Cin
7. Cout, PCin, END

In the case of conditional branch instruction, there are two different possibilities during the execute phase. If the condition is not satisfied, then no action is required during execute phase. If the condition is satisfied, then action similar to unconditional branch is required.



6.5 Multiple-bus Processor

The single-bus organization (Fig. 6.14) yields a simple but a slow speed processor. A large number of control sequences are required for each instruction cycle since there is only one bus. In order to speed up processing, multiple internal buses can be placed inside a processor. This enables parallel communication using the multiple paths provided by the buses. The datapaths of two such buses have been presented in chapter 4 (Fig. 4.9 and Fig. 4.10). Consider Fig. 6.25 showing three-bus datapath. The features of this datapath are as follows:

1. **Register file:** This has one input and two output ports. In a single clock cycle, one register write and two register read operations can be performed simultaneously. The A bus and B bus receive data from read operation and the C bus supplies data for write operation.
2. **ALU source/destination:** During arithmetic or logic operations, the A bus and B bus supply source operands to the ALU. The C bus receives the result from the ALU.
3. **ALU input/output registers:** Compared to the single-bus organization, the T register at the input of ALU and the C register at the output of ALU are eliminated since they are unnecessary.

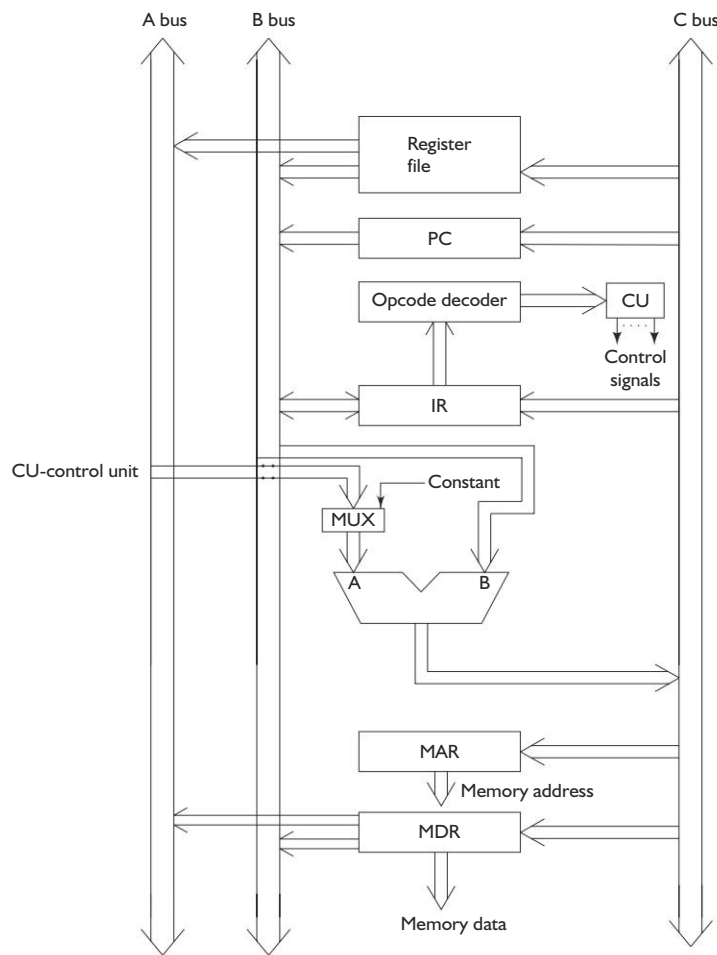


Fig. 6.25 Three bus based datapath

Example 6.1: Show the microoperation sequence and control sequence for the three bus processor for the instruction ADD <R1>, R2.

Microoperation sequence:

1. B bus $\leftarrow$ PC
2. C bus $\leftarrow$ B bus
3. MR
4. Loop MFC
5. B bus $\leftarrow$ MDR
6. C bus $\leftarrow$ B bus, IR $\leftarrow$ C bus
7. B bus $\leftarrow$ R1, C bus $\leftarrow$ B bus
8. MAR $\leftarrow$ C bus
9. MR
10. Loop MFC
11. B bus $\leftarrow$ MDR
12. A bus $\leftarrow$ R2
13. Add
14. C bus $\leftarrow$ Adder O/P
15. R2 $\leftarrow$ C bus

The step 2 microoperation of B bus to C bus transfer takes place through the adder.

Control sequence:

1. PCout, R = B, MARin, MR
2. Loop MFC
3. MDRout-B, R = B, IRin
4. R1out-B, R = B, MARin, MR
5. Loop MFC
6. MDRout-B, R2out-A, SEL MUX-A, Add
7. R2in, END

R = B means B bus contents are routed through the adder to the C bus. MDRout-B means the MDR contents enter the B bus. Similarly, R1out-B means the R1 contents enter the B bus. SEL MUX-A means enable A input of MUX (i.e. A bus).

6.6 Control Unit Design Options

The control unit is designed with two different capabilities:

1. The set of control signals is unique for each instruction. This intelligence is 'built' or 'stored' inside the control unit. The control unit supplies a set of control signals corresponding to the current opcode.
2. The control signals for an instruction are needed in a proper sequence. This information is also a part of the intelligence mentioned in point 1.

Basically the control unit converts each machine language instruction into a series of control signals which activate the control points in datapath. Figure 6.26 shows the input and output of the control unit.

There are two ways to design a control unit: hardwired or microprogrammed. In a *hardwired control unit*, the digital circuits generate the control signals whereas in a *microprogrammed control unit*, the control signals are stored as bit patterns in a read only memory, inside the control unit.

The hardwired control unit is the conventional design technique whereas microprogrammed control unit is relatively a modern technique. The following sections gives an in-depth view of these two types of control units.

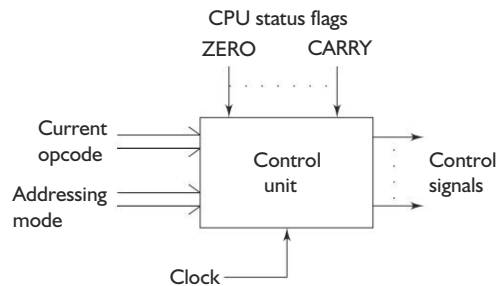


Fig. 6.26 Input and output of control unit

6.7 Hardwired Control Unit

A hardwired control unit consist of a collection of combinational circuits to generate various control signals. Figure 6.27 shows an overall block diagram of the hardwired control unit. The clock is a periodic signal that gives timing reference. The opcode identifies the

current instruction. The CPU status flags indicate the present status of CPU such as overflow, carry etc. In addition, special situations like 'waiting for MFC' are also included in these. Using these three types of inputs, the combinational circuits in the control unit generate the relevant control signals. The principle of operation of the hardwired control unit is shown in Fig. 6.28. Each dot represents a circuit that generates one or more control signals. Each vertical line represents an instruction. Each horizontal line represents a timing pulse train.

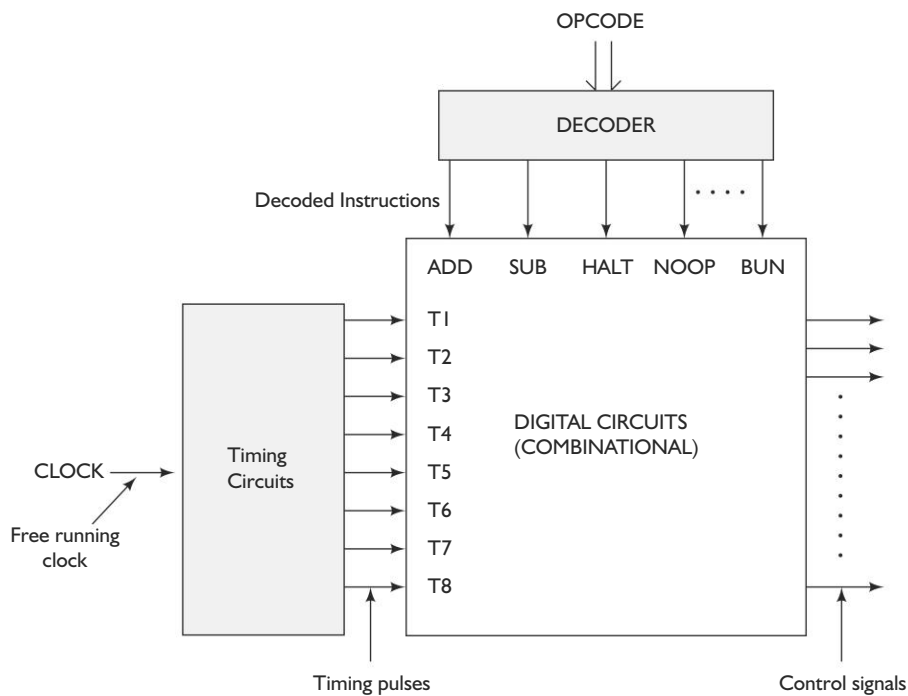
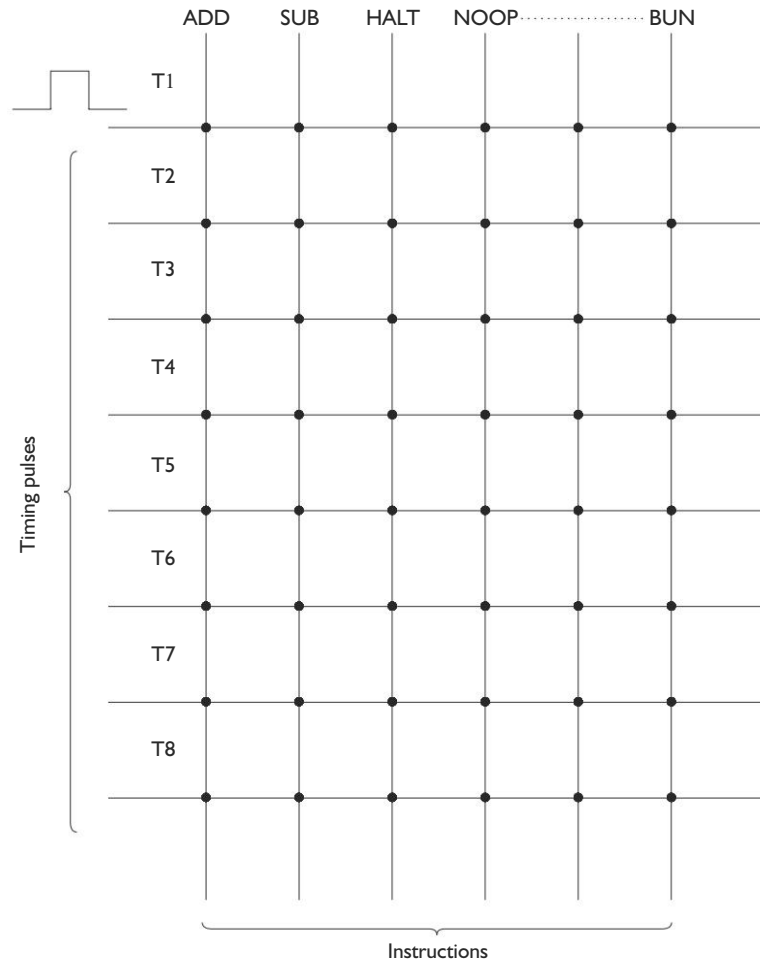


Fig. 6.27 Block diagram for hardwired control unit

The timing pulses T1, T2, T3 etc. have definite time delays. They are used as time reference signals. For a given instruction, the required microoperations are time sequenced using the timing pulses. For example, for LDA instruction, if reading from memory is done in T1, storing in accumulator can be done in T2. The signal combination of 'LDA' and 'T1'



Each dot represents a circuit that generates one or more control signals

Fig. 6.28 Basic principle of hardwired control unit

can be used as the control signal for reading from the memory. Another signal combination of 'LDA' and 'T2' can be used as the control signal for storing in the accumulator. Figure 6.29 shows how two gates generate these control signals during the execution of the LDA instruction. This concept is used for generating all the control signals. For generating some of the control signals, status of the CPU flags is used. For example, for JZ (Jump, if the accumulator is zero) instruction, the zero flag is used along with T1 and JZ as shown in Fig. 6.30. The control signal is generated, if the zero flag is 1.

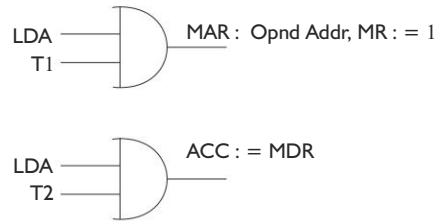


Fig. 6.29 Control signals generation for LDA instruction

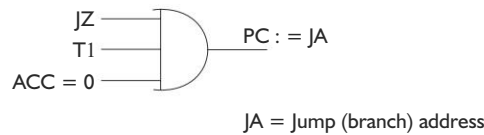


Fig. 6.30 Control signal generation for JZ instruction (Jump if accumulator is zero)

The hardwired control unit has circuits for the following:

1. Identifying the instruction from the opcode and generating relevant signals such as ADD, NOOP, HALT etc. At a given time, one of the instruction signals corresponding to the opcode is active and others are inactive.
2. Maintaining time reference by generating a sequence of timing pulses.
3. Generating a set of control signals corresponding to the microoperations to be done for the current instruction at proper time intervals.
4. After completing the execution of one instruction, generation of control signals necessary for fetching the next instruction.
5. Handling special sequences such as reset sequence, interrupt sequence and abnormal situation handling.

Figure 6.31 gives a detailed block diagram for the hardwired control unit. The opcode is the main input to the control unit. It is an input to the opcode decoder. A decoder is a combinational circuit with n inputs and 2^n outputs. At a given time, only one of the outputs is active. The active output identifies the *input signals combination*. The opcode decoder analyzes the opcode pattern and generates one output signal corresponding to the current opcode. If the currently fetched instruction is ADD, then the ADD output of the opcode decoder is active and others are inactive. The ADD signal goes to several gates in the combinational circuits. The clock signal is a free running clock. As long as the computer is powered-on, the clock is generated continuously irrespective of what is happening inside the computer. Even when the CPU is halted, the clock signal is generated.

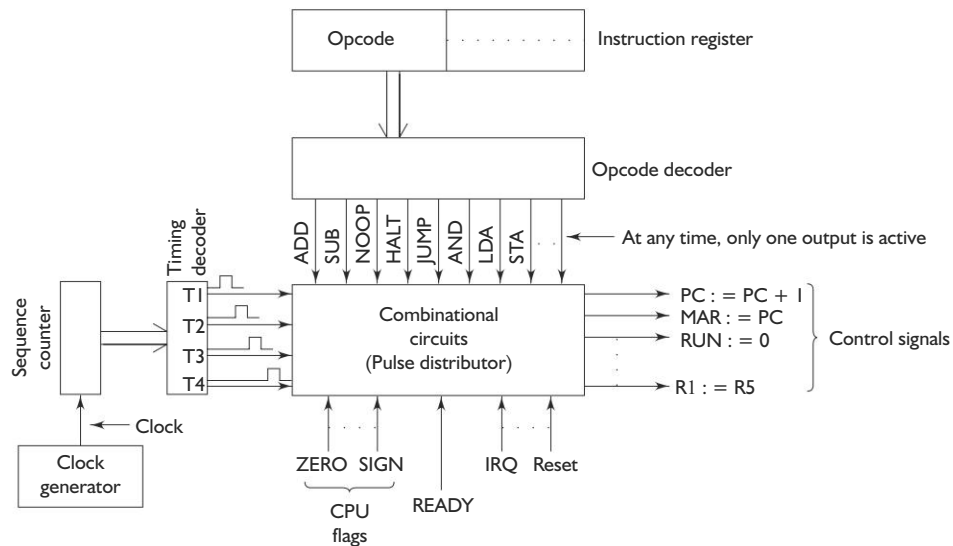


Fig 6.31 Hardwired control unit organization

The sequence counter is basically an up-counter. Its contents are incremented for every clock signal. On reaching its final count value, it is reset to zero by the next clock. At a given instant, its contents indicate the elapsed time from start in terms of the number of clock pulses. If we have a two bit sequence counter, the count status is 00, 01, 10, 11, 00 etc. The sequence counter's output is decoded by the timing decoder. At any time, one of the output of timing decoder is active. Figure 6.32 shows the waveforms for the timing decoder output. Each of the timing signals (T1, T2, T3 and T4) is a periodic waveform since their

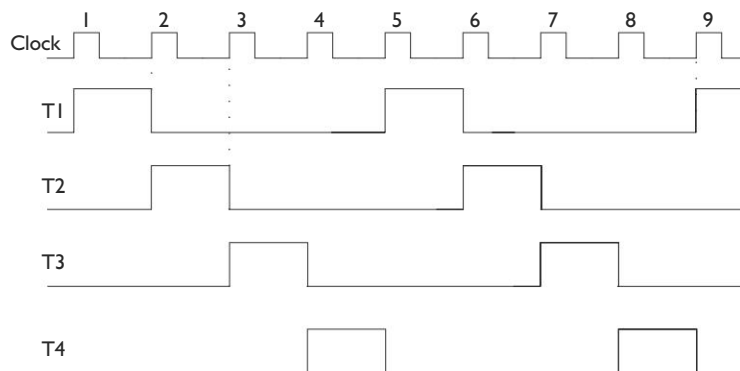
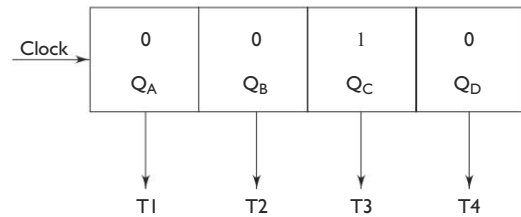


Fig. 6.32 Generation of timing signals



The pattern shown (0010) is before the arrival of the clock pulse. After the clock pulse, the pattern changes to 0001.

Fig. 6.33 Ring counter

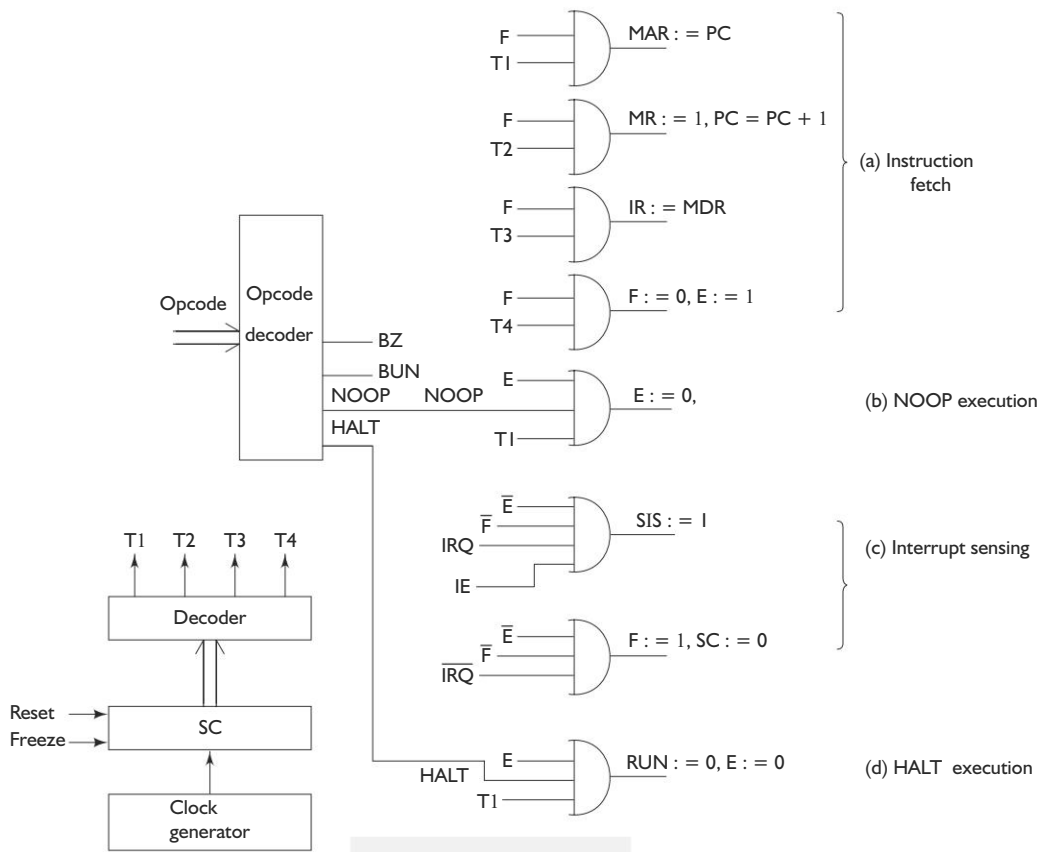


Fig. 6.34 Hardwired control unit: control signals generation

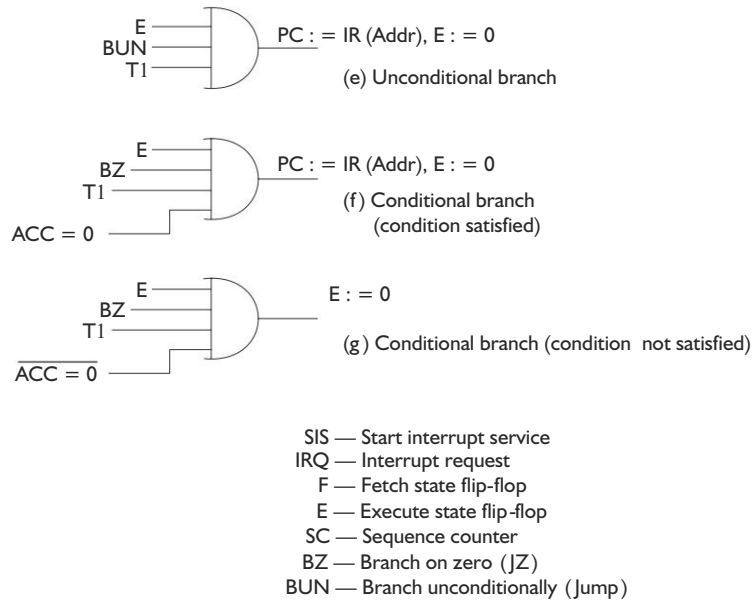


Fig. 6.34 Hardwired control unit: control signals generation

shapes are repetitive. It is possible to use a ring counter replacing both the sequence counter and timing generator. A ring counter (Fig. 6.33) has a rotating '1' which moves from one bit position to next when the clock signal comes. Figure 6.34 (a)–(g) shows some sample cases of combinational circuits needed for generating various control signals. The F and E flip-flops indicate whether the CPU is in instruction fetch phase or instruction execution phase of an instruction cycle. If both are reset, it means that the processor is not running

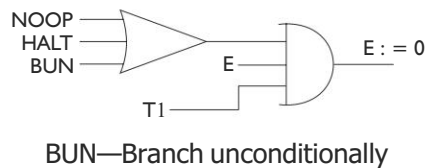


Fig. 6.35 Minimizing gates for control signals generation

(i.e. instruction cycles are not being performed). A careful analysis of Fig. 6.34 indicates that certain gates can be eliminated by combining the circuits of different instructions. Figure 6.35 explains this with an example. An experienced designer may directly arrive at the reduced configuration. Figure 6.36 shows the circuit for program counter control.

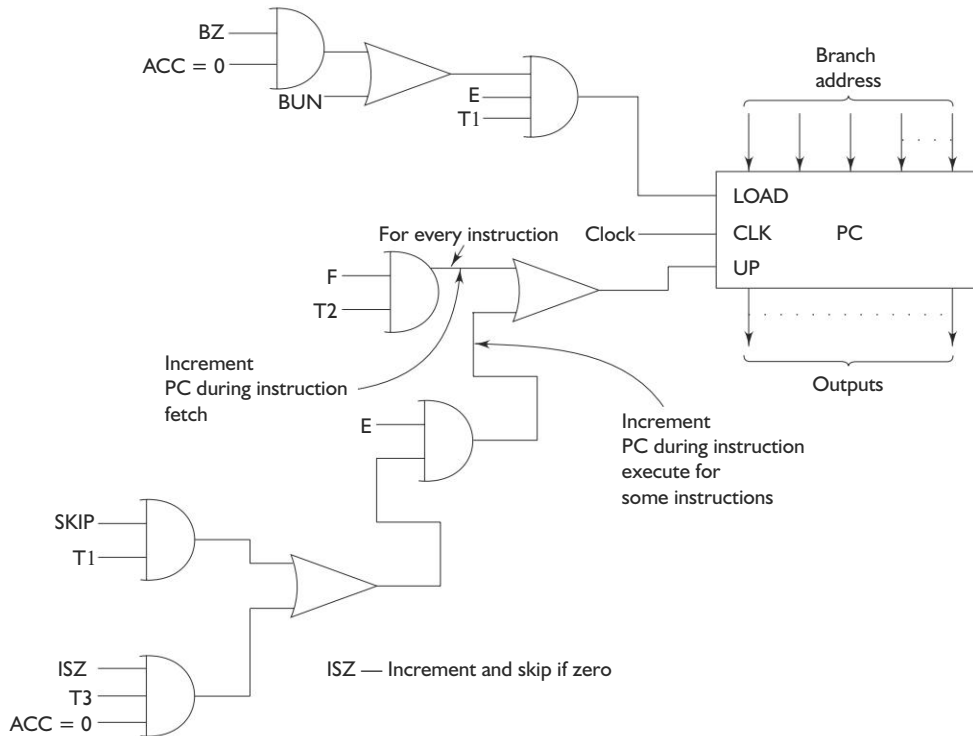


Fig. 6.36 Program counter control (reset vector entry not shown)

6.7.1 Synchronous and Asynchronous Control Unit

Consider the case of two microoperations X and Y to be executed one after the other: X first and Y next, after the completion of X. A synchronous control unit (Fig. 6.37) works purely with the clock timing for progressing from one microoperation to next. With the arrival of first clock signal, the control unit generates control signal for X and when the second clock arrives, it assumes that the previous microoperation (X) has been completed and generates control signal for Y. If in case, the time taken by the microoperation X is more than one clock period but less than two clock periods, the control unit does not generate the control signal for Y in the next clock. Only in the third clock, it generates the control signal for Y. In short, the control unit does not receive any feedback from the datapath about the completion of the microoperations. While designing the synchronous control unit, proper 'idling' is done by appropriate circuits. An asynchronous control unit (Fig. 6.38) does not

waste time. It waits for the 'completed' signal from the datapath for X and then starts next microoperation Y on receiving the 'completed' signal without wasting time. Though the waiting time is reduced, design of asynchronous control unit is complex and one is likely to make mistakes if sufficient care is not taken. Troubleshooting a system with synchronous control unit is easier than a system with asynchronous control unit.

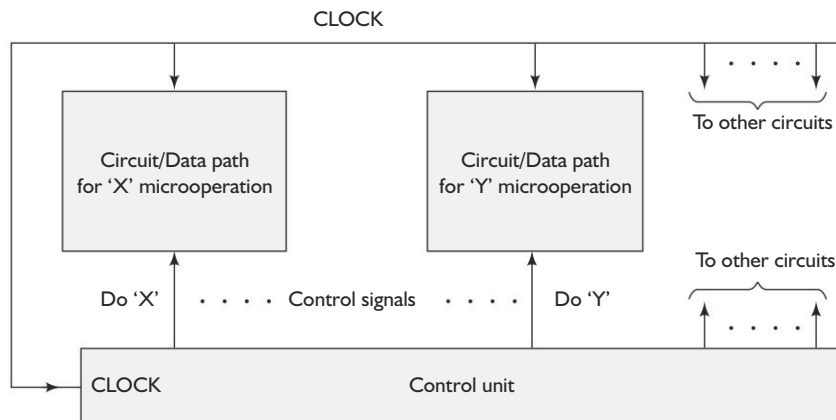


Fig. 6.37 Principle of synchronous control unit

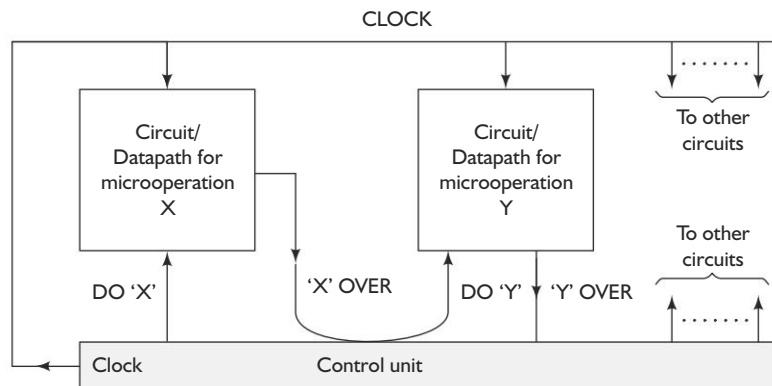


Fig. 6.38 Principle of asynchronous control unit

6.7.2 Advantages of Hardwired Control Unit

A hardwired control unit works fast. The combinational circuits generate the control signals based on the input signals status. As soon as the required input signal combination occurs, immediately the output (control signal) is generated by a combinational circuit. The delay between the output generation to input availability depends on the number of gates in the circuit path and propagation delay of each gate in the path. If there are two gate levels and each gate has 5 ns propagation delay, the time taken for the control signal generation is 10 ns.

6.7.3 Disadvantages of Hardwired Control Unit

1. If the CPU has a large number of control points, the control unit design becomes very complex. It is tedious to design the pulse distributor circuitry (combinational circuits) since several signal combinations have to be kept track of during designing.
2. The design does not give any flexibility. If any modification is required, it is extremely difficult to make the correction. Design modification becomes necessary under the following situations:
 - (a) There is a design mistake in the original design.
 - (b) A new feature (e.g. a new instruction) is to be added to an existing design.
 - (c) A new hardware component (e.g. memory) of higher speed is available, which will improve the performance.

The hardwired control unit is popularly known as random logic, in view of usage of too many gates in the design.



6.8 Microprogrammed Control Unit

Microprogramming is a modern concept used for designing a control unit. It can be used for designing control logic for any digital system. Some common applications are I/O controllers such as disk controller, and peripheral devices such as printer and hard disk drive. The philosophy of microprogramming is based on 'reading stored control patterns'.

The microoperations are executed when the corresponding control signals are made active. Each instruction needs a specific set of microoperations (Fig. 6.39) in an order (time sequence). The control signals needed for an instruction and the time sequence can be stored in the memory known as control memory. By fetching the control memory words one-by-one, the control signals can be generated. The control memory is a read only

memory. Any common activity occurring in control unit such as interrupt handling, instruction fetch etc. can be translated into control memory words. The control unit can only read from the control memory but cannot modify the contents. In a simplest form,

1	2	3	4	5	6	7	8	9	10	11	12	13	14
PC _{in}	PC _{out}	MAR _{in}	MAR _{out}	MDR _{out}	IR _{in}	IR _{out}	R _{1in}	R _{1out}	R _{2in}	R _{2out}	R _{3in}	R _{3out}	T _{in}
15	16	17	18	19	20	21	22	23	24	25	26		
C _{in}	C _{out}	MUX SEL	PC+1	MFC	MR	MW	SP1 _{in}	SP1 _{out}	SP2 _{in}	SP2 _{out}	EOE		

Fig. 6.39 Control word format for single-bus datapath

each bit in the control memory word can be reserved for a control signal; thus, separate bits for all control signals can be used. If the bit is 1, the corresponding control signal is activated. If the bit is 0, the control signal is not activated. When a control memory word is fetched from the control memory, some bits may be 1 and others 0. The control signals corresponding to '1' bits are generated. Thus, multiple microoperations are executed simultaneously. Each control memory word is known as a microinstruction. For the single-bus processor shown in Fig.6.14, let us develop a microprogram for ADD instruction. First, we have to identify the total number of control points so that the length of the control memory word can be decided. Fig. 6.39 shows a 26-bit control word for the single-bus processor. The following are the assumptions made:

1. Only 3 GPRs (R1, R2 and R3) are present
2. The scratch pad contains only two registers: SP1 and SP2.
3. Incrementing PC is a control point activated indicated by PC+1
4. MAR_{in} is not included since it is irrelevant
5. End of execute microprogram is indicated by EOE
6. Adder is combinational. It has only addition capability.

The reader needs to be very careful in writing microinstructions manually. However, there are powerful software tools such as cross assemblers for developing micropgrams which simplify the task. To give another example, another simple control memory word is shown in Fig. 6.40 for the main frame CPU (shown in Fig.4.23) covered in section 4.11. Consider Fig. 6.41 that gives fetch microprogram consisting of three microinstructions.

The First Microinstruction results in generating the following control signals, simultaneously:

1. $MAR := PC$ PC contents are transferred to MAR
2. $MR := 1$ Memory read signal is set

The Second Microinstruction results in generating the following two control signals, simultaneously:

1. $IR := MDR$ Memory data register contents are transferred to IR
2. $PC := PC + 1$ The PC contents are incremented

The Third Microinstruction clears F flip-flop and sets E flip-flop.

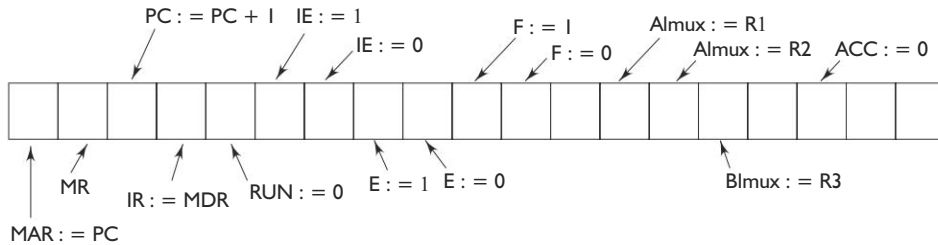


Fig. 6.40 Simple control memory word format

```

1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0

```

Fig. 6.41 Fetch microprogram

The three microinstructions collectively perform instruction fetch phase. The control unit fetches these microinstructions from control memory one-by-one. As soon as a microinstruction is loaded in the microinstruction register, the control signals which are indicated as '1' in the microinstruction register, are activated. As a result, the relevant microoperations are performed.

6.8.1 Issues in Microprogrammed Control Unit

Figure 6.42 shows the basic requirement of a microprogrammed control unit. The Control Memory Address Register (CMAR) is also known as a microprogram counter. The CMDR is a control memory data register. It is also known as microinstruction register. The following issues have to be resolved in a microprogrammed CU:

1. There is a separate execute microprogram for each instruction such as—ADD, SUB, LDA, etc. After the instruction fetch is over, how does the execute microprogram gets control?
2. After the completion of any execute microprogram, how does control goes to fetch microprogram?
3. After the power-on, how does the control unit start the appropriate microprogram ?
4. Frequently, certain microinstructions have to be skipped and the control unit has to branch on the basis of certain status in the CPU. For example, the carry flag or zero flag status dictates the exact action required after the completion of an arithmetic operation. Similarly, the IE flag status decides the action path to be followed on sensing the interrupt request. Hence, we need a mechanism for conditional branching.
5. Assigning a bit to each control signal leads to a long width of the control memory word. This makes it expensive. Hence, we need techniques to reduce the length of microinstruction.

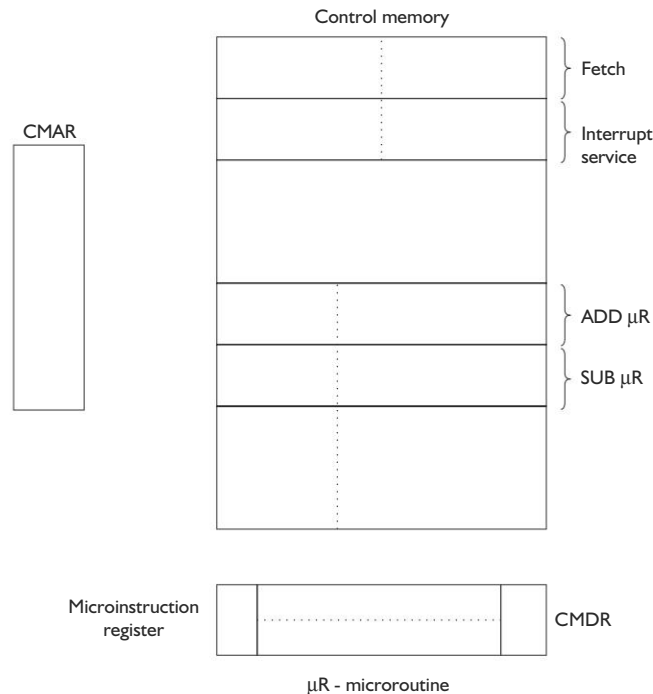


Fig. 6.42 Microprograms and control memory

6. Many micro operations of certain instructions are common. For example, the actions required for ADDB and SUBB instructions differ only in one aspect. The ADD instruction uses B whereas the SUB instruction uses complement of B. Otherwise, all the micro instructions are same including addition. Having totally separate execute microprograms for each instruction, increases the size (number of locations) of the control memory. We need some technique for combining such microinstructions. The ADD and SUB instruction can have a common execute microprogram and one should handle both these requirements by a single microroutine so as to reduce the control memory capacity.

Figure 6.43(a) shows a typical organization of a microprogrammed control unit taking into account practical issues. Figure 6.43(b) shows the generation of next microinstruction address.

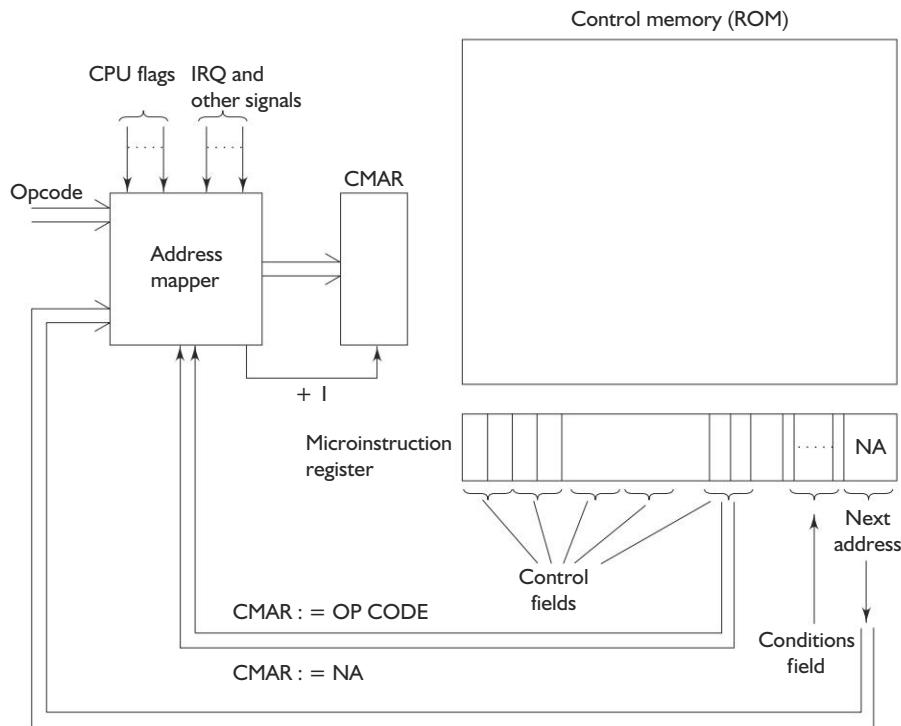


Fig. 6.43(a) Operation of microprogrammed control unit

6.8.2 Reducing Microinstruction Word Length

Certain microoperations (and hence their control signals) are mutually exclusive. For example, if we consider the following microoperations related to program counter, we will

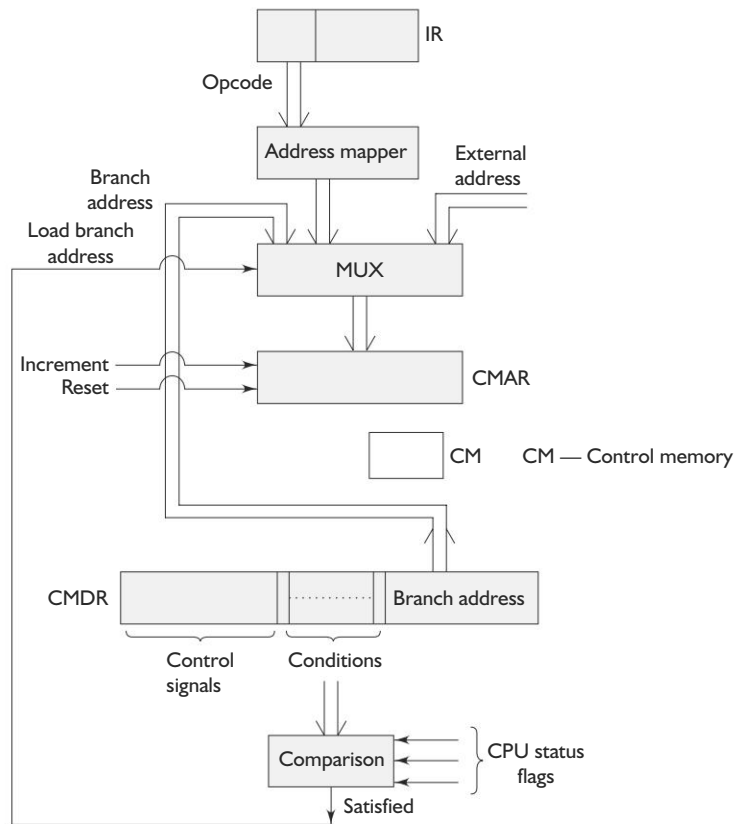


Fig. 6.43(b) Sequencer logic

see that in a given time, only one of them is executed depending on the condition of the instruction cycle.

1. $PC := PC + 1$
2. $PC := MDR$
3. $PC := EA/IR$ (Jump instruction)

Instead of assigning 4 bits in the microinstruction word, we can have 2 bits with encoded information. This results in saving of 2 bits. At the same time, a 2–4 bit decoder is used to decode a 2-bit field and give 4 output which correspond to the 4 control signals. We reduce the length of the control word, at the cost of adding a decoder circuit.

6.8.3 Horizontal and Vertical Microprogramming

Figures 6.44, 6.45 and 6.46 show three different ways of indicating control signal patterns. Having an individual bit for each control signal in the microinstruction format is known as a horizontal microinstruction, as shown in Fig. 6.44. Each bit activates one control signal. Several control signals can be simultaneously generated by a single microinstruction. The length of the microinstruction is increased (30–120 bits). A variation divides the microinstruction into multiple fields, as shown in Figs. 6.45(a) and (b). Each group encodes

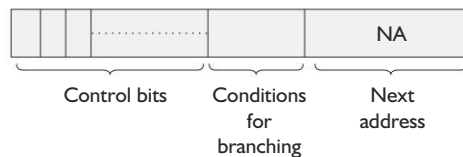


Fig. 6.44 Pure horizontal microinstruction

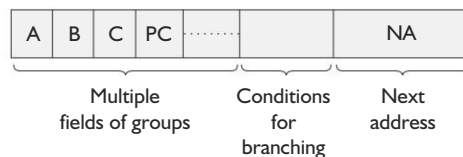


Fig. 6.45(a) Horizontal microinstruction with multiple groups

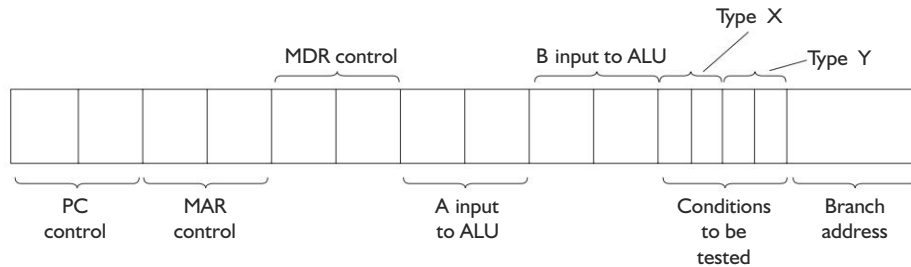


Fig. 6.45(b) Grouping control bits into fields

mutually exclusive control signals (microoperations). This reduces the length of the microinstruction. But a decoder circuit is needed for each field to decode the pattern and identify the control signal. The total number of simultaneous control signals (and hence, microoperations) cannot exceed the number of fields. As a result, the number of microinstructions needed in a routine is more compared to a pure horizontal microprogram. This causes additional delay as more number of microinstructions are fetched from the control memory. In a vertical microinstruction (Fig. 6.46), a single field can produce an encoded sequence. The vertical microprogram technique takes more time for generating the control signals due to the decoding time and also more microinstructions are needed. But the overall cost is less, since the microinstructions are small in size (12–30 bits). The horizontal microprogram releases faster control signals but the control memory size is huge due to increased word length.

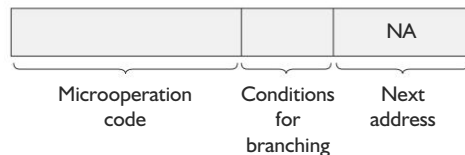


Fig. 6.46 Pure vertical microinstruction

6.8.4 Giving Control to Execute Microprograms

At the end of fetch microprogram, execution of the following microinstruction provides control to the execute microprogram.

CMAR := OCR

Assume we have 8 bits in the opcode, so we need 256 execute microprograms each starting from different addresses. Hence at the end of fetch microprogram, the control unit causes a jump to the address where the execute microprogram for that opcode begins. This can be done in one of the following two ways:

Method 1. Store the start addresses of all execute microprograms as jump address field (next microinstruction address) in continuous locations in control memory as shown in Fig. 6.47. Use a hardware circuit (mapper) whose input is opcode and output is the location containing the jump microinstructions (Fig. 6.48).

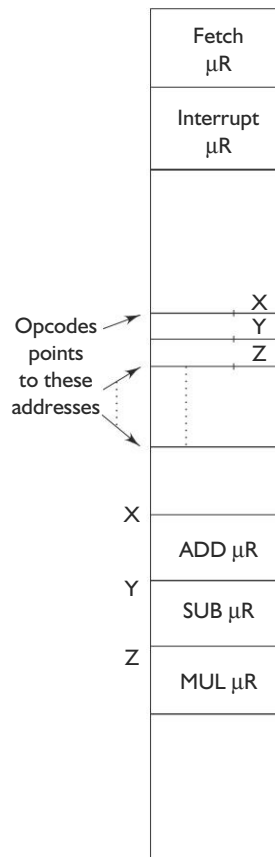


Fig. 6.47 Entering execute microroutines: indirect branching from OPCODE

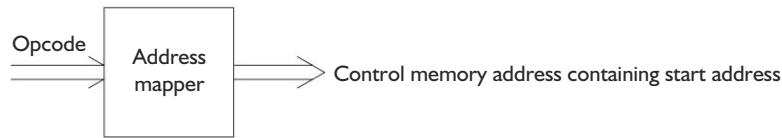


Fig. 6.48 Indirect entry to execute microprograms

Method 2. The last microinstruction in fetch microprogram should have a jump microinstruction pointing to the start address from where the execute microprogram begins. Suppose we chose the opcodes in such a way that the opcode itself points to the start address of the execute microprogram, then the last microinstruction in fetch microprogram can simply include a microoperation $CMAR := OCR$. Figure 6.49 and 6.50 display this mechanism. Figure 6.51 elaborates the derivation of a 12-bit control memory address from a 4-bit opcode.

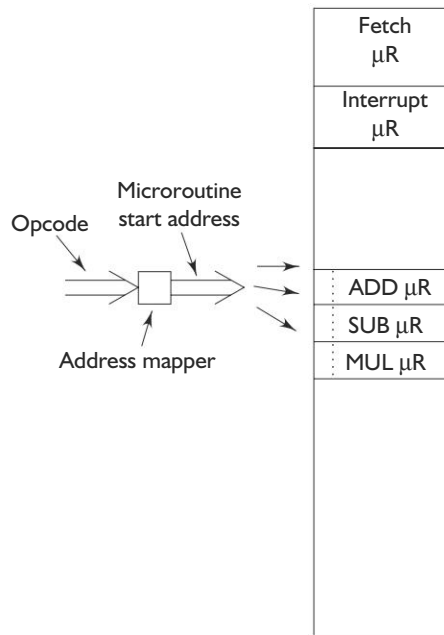


Fig. 6.49 Entering execute microroutines: converting opcode as start address

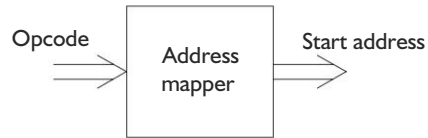
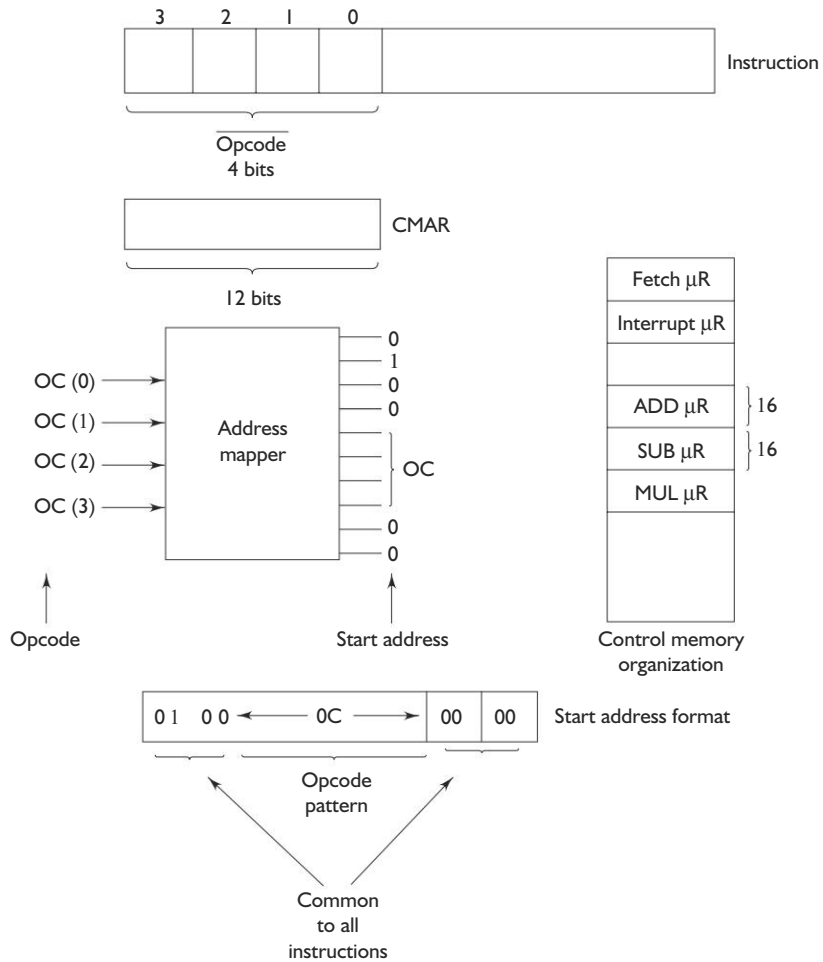


Fig. 6.50 Direct entry to execute microprograms



As per this format, each 'execute' microroutine can have a maximum of 16 microinstructions,

Fig. 6.51 Deriving start address for execute microroutine

6.8.5 Unconditional Branching

The microinstruction gives next microinstruction address. One of the control bits generates the control signal $\text{CMAR} := \text{NA}$. Figure 6.52 illustrates this.

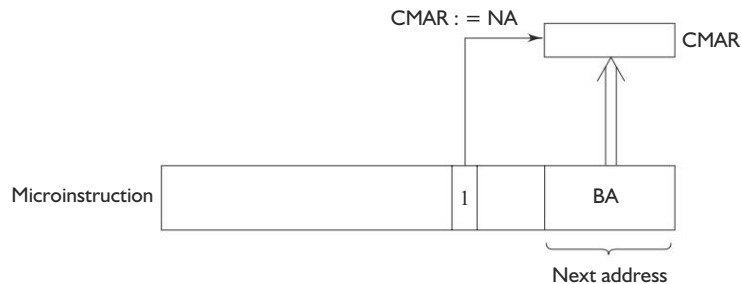


Fig. 6.52 Unconditional branching

6.8.6 Conditional Branching

Each bit in the horizontal microinstruction corresponds to one control signal. The branch condition field specifies the condition to be tested by the current microinstruction. Some of the common conditions tested are:

1. *Zero flag*: accumulator contents zero
2. *Overflow flag*: for result overflow
3. *Carry flag*
4. *Interrupt Enable flag*
5. *Sign flag*

If the condition tested is successful, the branch address contains the next microinstruction to be executed, otherwise the physically next address contains microinstruction.

6.8.8 Advantages of Microprogrammed Control Unit

1. *The design of microprogram control unit is less complex* as compared to the hardwired control unit. Often people say that microprogramming is a systematic way of designing the control unit. However, it should be noted that even the hardwired CU is designed systematically, but it is more complex. The design of a hardwired CU needs the knowledge of digital electronics, whereas the design of a microprogrammed CU can be done without it.
2. *The microprogrammed control unit is flexible* since modification is easy as it involves simply changing the contents of control memory. This simplifies design correction, design enhancement and modification.
3. *The meaning of a given CPU's instruction set can be easily modified* by changing the microprograms without affecting the datapath. This facility is used to emulate another system using the present system. Emulation of a future system under the planning stage is also possible by modifying the microprograms of an existing system. This helps in evaluating the performance of a system in advance.
4. *The debugging and maintenance of a microprogrammed CPU is easy.* The microinstructions can be executed one-by-one in a diagnostic mode, simultaneously monitoring the contents of registers, counters, flags etc. Similarly, autodiagnosics is possible with the help of microdiagnostics.

6.8.9 Disadvantages of Microprogrammed Control Unit

1. *A microprogrammed CU is slow.* The microinstructions are stored in the control memory. Fetching them takes time. Since one instruction cycle is covered by several (varying from 3 to 20) microinstructions, total instruction cycle of a microprogrammed CU is more.
2. *For a small CPU with very limited hardware resources,* a microprogrammed CU is expensive as compared to a hardwired CU.

6.8.10 Static and Dynamic Decoding

Reduction of microinstruction length results in heavy cost saving. One of the techniques adopted is dynamic decoding. The discussion so far covered Static Decoding (Fig. 6.54). A field has only one meaning. Figure 6.55 shows a microinstruction with static

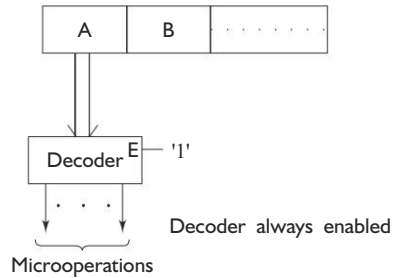
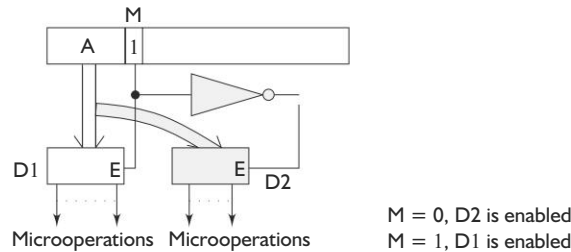


Fig. 6.54 Static decoding

decoding. In dynamic decoding, the same field may mean different control signals depending on some other control field (ESC). Figure 6.56 shows the concept of dynamic decoding with dual decoders for each field. The field A has two sets of encoding. In a given microinstruction, the M bit indicates the set needed currently. Accordingly, the corresponding decoder is enabled. With the same number of bits (plus ESC field), twice the microoperations can be covered by dynamic decoding.

CA	CB	CC	AA	CS	PC		NA
↑ Source for A-input of ALU	↑ Source for B-input of ALU	↑ Destination for ALU output	↑ ALU control	↑ Shifter control	↑ Program counter control		↑ Conditons tested
CA field	meaning				PC pattern	Meaning	
0 0 0	NIL				0 0	NIL	
0 0 1	R1				0 1	Reset vector	
0 1 0	R2				1 0	Accumulator	
0 1 1	R1				1 1	Jump address (IR)	
1 0 0	R2						

Fig. 6.55 Sample format of a practical microinstruction

**Fig. 6.56** Dynamic decoding

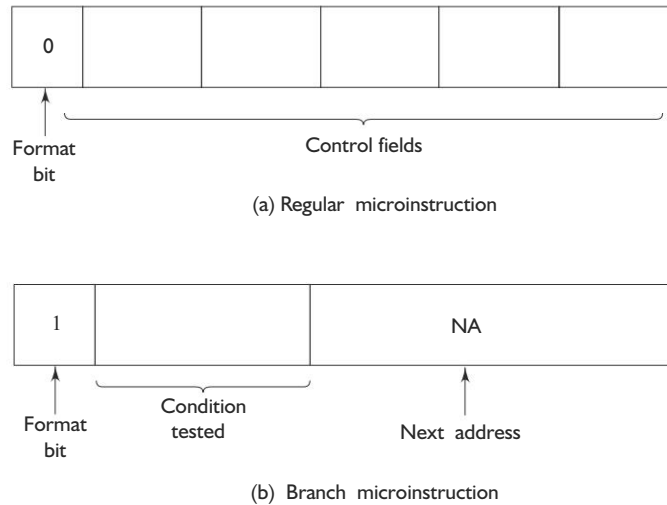
6.8.11 Dual Format Microinstruction

A popular mechanism for specifying the next microinstruction address in case of branch is having a separate microinstruction. A regular microinstruction shown in Fig. 6.57, part (a) does not have address field. All fields specify control signals. All decoders are enabled, with the format bit being 0. In this case, the next microinstruction is the physically next microinstruction. For branching, a separate microinstruction of Fig. 6.57, (b) type is used. The format bit being 1 disables all decoders of control fields. In this format, except specifying the branch condition and the branch microinstruction address, no other microoperation is specified.

6.8.12 Emit Field

Generation of constants is necessary for many reasons:

1. For address of reserved memory locations such as interrupt service routine start address.
2. All 1's data for ALU.
3. Mask patterns for opcode and mode field analysis.

**Fig. 6.57** Dual format microinstruction

A common method is using a separate field to store the constants. This field is known as emit field.

6.8.13 Nanoprogramming

The nanoprogramming is a two level microprogramming (Fig. 6.58). Each location in the microprogram memory (level 1) leads to a nanoprogram (level 2) in control memory. The level 1 control memory word consists of two fields:

1. Next nanomemory address
2. Next microprogram address

The nanomprogram memory gives control signals. The advantage of two level control memory is reduction in control memory size.

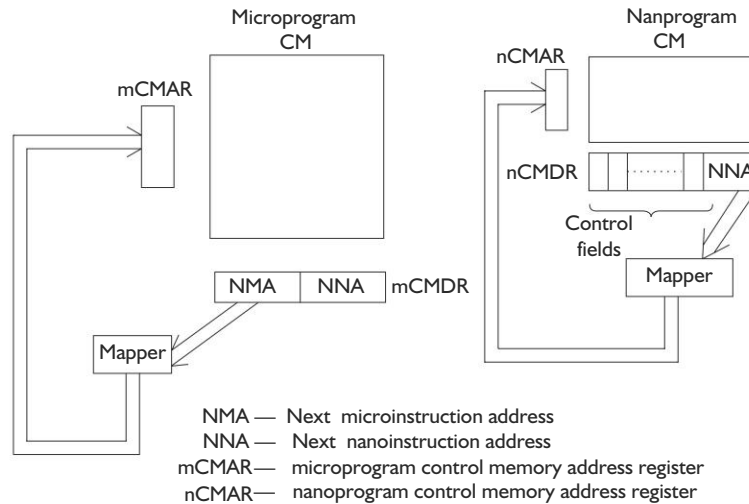


Fig. 6.58 Nanoprogramming concept

SUMMARY

The control unit is the ‘brain centre’ of a computer. The overall synchronization and coordination of the hardware units is done by the control signals supplied by the control unit. Each control signal activates a control point resulting in the execution of a microoperation. Functionally, the control unit is part of the processor but its responsibility includes generation of control signals to circuits inside the processor such as ALU, registers etc as well as units outside the processor such as main memory, I/O controllers etc.

The control unit is designed diligently for performing the following functions:

- Reset sequence
- Instruction fetch sequence
- Instruction execution
- Interrupt handling
- Bus control
- Abnormal situation handling.

Instruction fetch is a common function of all types of instructions and the action taken by the control unit is similar for all these. Operands fetch and instruction execution differs for various opcodes. The control unit analyzes the opcode pattern and the addressing mode and accordingly, takes appropriate action (generating relevant control signals).

A synchronous control unit fixes different time slots for various microoperations performed by the hardware logics/components. It does not get any feedback from the logic units about the completion of the microoperations. All units synchronize themselves with the clock signal issued by the control unit for timing information. An asynchronous control unit waits for the feedback/acknowledgment from individual logics for every microoperation. An asynchronous control unit can operate faster than a synchronous control unit. But a synchronous control unit can be designed easily and the debugging of a hardware fault in the computer is easier.

The early processors' control units belong to the type known as hardwired control unit or random logic. It generates different control signals by the hardware circuits. It works faster but the design becomes complex for a large processor. Design modifications in a hardwired control unit is a tedious procedure since it is not flexible.

A microprogrammed control unit follows 'stored program' concept for generating the control signals. It has a read only memory inside the control unit for storing the control signal patterns. The control signals and their sequence for different instruction execution and other actions are stored as microinstructions. The microprogrammed control unit is slower because of fetching microinstructions from ROM. But the design is easier and the modification is possible. It is called firmware as it is neither hard nor soft.

The horizontal microprogram results in quick execution but occupy more space in the control memory whereas the vertical microprogram reduce the cost by minimizing control memory space but it is slower.

REVIEW QUESTIONS

1. NOOP instruction requires nothing (no action) to be performed for the instruction. This is true for the macrooperation level, but false for the microoperation level. The control unit must perform one microoperation which is necessary for any instruction. Identify the essential microoperation which is performed even for NOOP instruction.
2. Pick the incorrect (faulty) RTL statement and indicate the problem:
 - (a) $PC := MAR, PC := PC + 1$
 - (b) $MR := 1, PC := PC + 1$

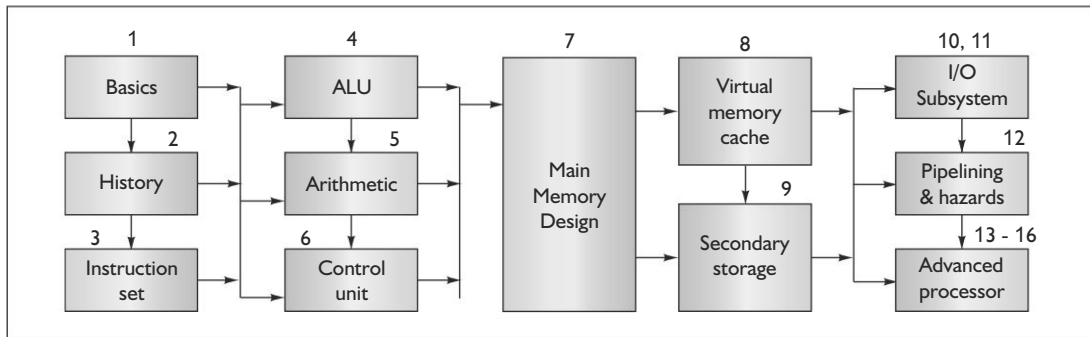
EXERCISES

1. A company assigns two different design teams to develop a microprogrammed control unit for a CPU. The team A gave the following proposal:
 - (a) control memory size: 256 locations of 80 bit each
 - (b) microinstruction type: horizontal
 - (c) no. of bits for branch address: 8The team B proposed nanoprogramming (a two level microprogramming). The nanoprogram memory size is 64 locations of 300 bits each. Which team's proposal is better from the point of minimizing the control memory size?
2. A microprogrammed CPU has 1 K words in control memory. Each instruction needs 8 microinstructions. Assuming that the opcode in the macroinstruction is of 5 bits length, propose a mapping scheme to generate control memory address for the opcode.
3. A control unit's microinstruction has 9 bits for specifying the microoperations. Assume that it is divided into two fields: one having 5 bits and the other with 4 bits. How many different microoperations are possible in the CPU?
4. Write down the microoperations needed for the following instruction:
EXCHANGE R1, R3
5. A CPU's microinstruction format has five separate control fields. The number of microoperations in each field are as follows:
 $F1 = 4$, $F2 = 4$, $F3 = 3$, $F4 = 12$, $F5 = 21$.
 - (a) What is the total length of the microinstruction needed to accommodate the five control fields?
 - (b) If pure horizontal microprogramming is followed without encoding, what will be the length of microinstruction?
6. A processor has following hardware configuration:
 - (a) No. of registers = 8
 - (b) ALU operations: arithmetic 8, logic 8
 - (c) Shifter : 4 operations
 - (d) Bus: single busDesign the microinstruction format for this CPU.
7. Using the control word format in Fig.4.39, write microprogram for the instruction $\text{ADD } \langle R1 \rangle, R2$ for which the control sequence is given in section 6.4.4 for a single-bus processor.

CHAPTER 7



MAIN MEMORY DESIGN



Chapter Flow Diagram

OBJECTIVES

- « *Memory types*
- « *Memory characteristics*
- « *Semiconductor main memory*
- « *Read/write memory*
- « *Read only memory*
- « *Main memory organization*
- « *Memory technology*

KEYWORDS

Random Access Memory	Double Data Rate Synchronous DRAM (DDR SDRAM)
Sequential Access Memory	Parallel Presence Detect (PPD)
Semirandom Access Memory	Serial Presence Detect (SPD)
Access Time	RAMBUS RAM
Cycle Time	RDRAM
Bandwidth	SIMM
Latency	DIMM
Static Memory	ROM
Dynamic Memory	PROM
Refresh	EPROM
Destructive Readout	EAROM
Volatile Memory	Pseudo Static RAM
Flash Memory	Memory Hierarchy
Interleaving	Cache memory
Asynchronous DRAM	Virtual memory
Synchronous Dynamic RAM (SDRAM)	
Fast Page Mode	

7.1 Introduction

The memory is used for storing information. The capacity of memory and speed are both important. Several memory types have been developed and used in practice. This chapter defines different memory types and concentrates on main memory design. The limitations of main memory and techniques of overcoming of these limitations are discussed in Chapter 8. The secondary memory is discussed in chapter 9.

7.2 Memory Parameters

There are four important parameters that are considered in choosing a memory:

- Capacity
- Speed
- Latency
- Bandwidth.

Capacity: Memory can be viewed as a storage unit containing l number of locations each of which stores w number of bits (Fig. 7.1). In other words, the memory has l addresses and its word length is w bits. The total capacity is $l \times w$ bits.

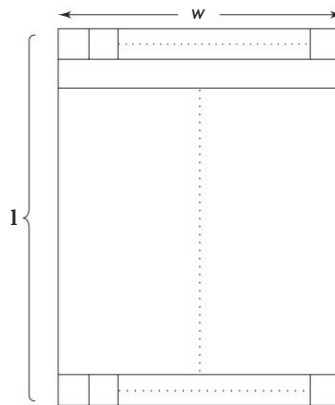


Fig. 7.1 A memory with l locations of each w bits

Speed: The memory speed is measured in terms of two parameters: *access time*, t_A and *cycle time*, t_C . To perform a read operation, first the address of the location is given to memory followed by the 'read' control signal (Fig. 7.2). The memory decodes the address, selects the

location and reads out the contents of the location. The *access time* is the time taken by the memory to complete a read operation from the instant of receiving the 'read' control

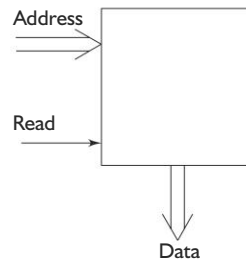


Fig. 7.2 Block diagram for read operation

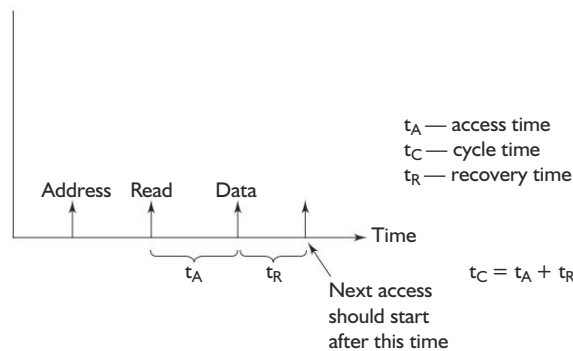


Fig. 7.3 Access time and cycle time

signal as shown in Fig 7.3. Generally access time for read and write is equal. Suppose two successive memory read operations have to be performed. After the first read operation, the information read from memory can be immediately used by CPU. However, the memory is still busy with some internal operation for some more time called recovery time, t_R . During this time, another memory access, read or write, can not be initiated. The cycle time is the total time including access time and recovery time: $t_C = t_A + t_R$. It is the minimum time interval necessary from the initiation of one memory operation to initiation of next operation. The value for recovery time varies with memory technology.

Latency: In some types of memory such as hard disk, the first access to any location (sector) has longer access time where as the successive locations have shorter access time. The latency is the access time for the first access in a series of accesses.

Bandwidth: The bandwidth is the data transfer rate by the memory; it is expressed as number of bytes per second.

7.3 Classification of Memory

The factors on which the memory is classified are—technology, access method, function and usage mode as shown in Fig. 7.4.

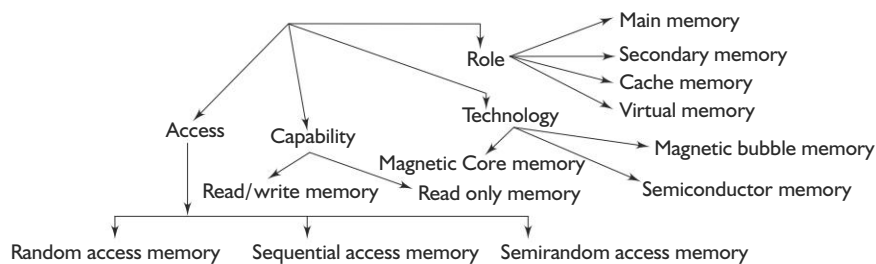


Fig. 7.4 Memory types

7.3.1 Access Method

The access method refers to the mechanism of accessing different locations in the memory. A Random Access Memory (RAM) is one which allows access to any location independent of other locations. In other words, access time is equal for all locations. Magnetic core memory and semiconductor memory are some of the examples for random access memory. For sequential access memory, the access time varies with the location. Magnetic tape is a typical example for sequential access memory. In a semirandom access memory, the selection of the location to be accessed involves two steps: one random access and one sequential access. Floppy disk and hard disk are examples of semirandom memory. It is also known as direct access memory.

7.3.2 Read/Write Capability

The Read Only Memory (ROM) allows only read operation by the CPU. No write operation is allowed. The read/write memory allows both read and write operation. In the computer industry, the semiconductor read/write memory is termed as RAM (instead of RWM) though technically, the semiconductor ROM is also random access memory.

7.3.3 Functional/Usage Modes

This classification is based on the role played by the memory in a computer. Basically, there are five different roles: internal registers, main memory, auxiliary memory, cache memory and virtual memory. The registers have a limited role of storing temporary information while running a program. But it has a very short access time. Certain registers are accessible only by the CPU whereas others are accessible by the program also.

7.3.3.1 Main Memory

The main memory is one from where the CPU fetches instructions during program execution. It is also known as program memory or primary memory. The main memory is of random access type. For a given CPU, the maximum main memory capacity is 2^n locations where n is the number of bits in the memory address. Practically, a computer has lesser physical memory than this due to cost considerations.

Physically the main memory can be integrated as a part of the processor IC as on-chip memory. Its size is usually limited because of IC cost and space considerations. Some portion of the main memory in present day computers is usually a read only memory, used to store permanent control programs and boot strap loader.

7.3.3.2 Auxiliary Memory

The auxiliary memory is also known as secondary memory. It is used to store large volumes of program and data. It is slower and cheaper than main memory. Floppy disk, hard disk, magnetic tape and optical disk are present day's commonly used secondary storage. The secondary storage contents can be shifted to the main memory whenever required (Fig. 7.5). The CPU accesses the secondary storage as a peripheral device. However, the operating system can manipulate the secondary storage as an extension of the main memory as seen in the case of the virtual memory concept. Most of the secondary memory are based on magnetic or optical storing principle. Additional coverage on the secondary memory are presented in Chapter 9.

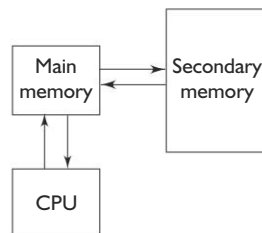


Fig. 7.5 Use of secondary memory

7.3.3.3 Cache Memory

The cache memory is an intermediate buffer between the CPU and the main memory. Its capacity is kept small since it is very expensive. Its speed is several times more than the main memory. Hence certain frequently required items are copied, in advance, from main memory to cache memory. When any of these is required by CPU, it is made available from cache memory instead of accessing the main memory. Effectively, a time consuming main memory access is replaced by a quick cache memory access. However, if the required item is not present in cache memory, then an access to main memory is unavoidable (Fig. 7.6).

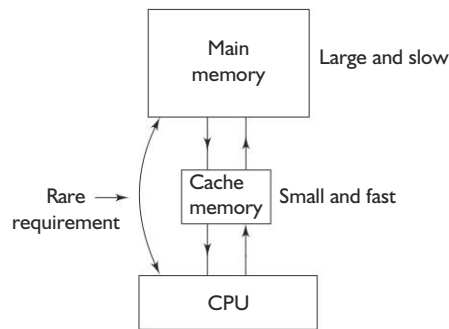


Fig. 7.6 Concept of cache memory

The cache memory can be physically integrated with processor IC as internal cache, also known as on-chip cache. Additional details about cache memory are provided in Chapter 8.

7.3.3.4 Virtual Memory

The virtual memory feature helps to execute a long program in a small physical memory. The OS (operating system) manages the program by keeping only part of it in main memory and using the secondary memory for keeping the full program. As and when a part is not available in main memory, it is brought from the secondary memory. The CPU hardware and the operating system collaborate in achieving this while the program is being run. The application programmer is not aware of the physical memory size. Additional details on virtual memory mechanism are given in Chapter 8.

7.3.4 Memory Technology

Earlier, magnetic core memory was widely used as the main memory in main frames and minicomputers, but it has been replaced by semiconductor memory since it is slow, consumes more power and space. Magnetic bubble memory is a serial memory, very slow and needs to be refreshed. Its control circuitry is also involved. Hence it did not play any role in computer even though it attracted designers in the initial days of announcement.

The core memory used the principle of magnetism for storing 1 or 0 on a magnetic core. A good property of core memory is its non-volatile nature. It retains its contents (1 or 0) even after the power is removed. A drawback of core memory is its non-destructive read-out nature. When supplying its contents during a read operation, it also loses the contents. Hence, a write-after-read is necessary to restore its contents. As a result, its cycle time becomes more.

Semiconductor memory's main advantages over core memory are its low power consumption, low access time, non-destructive read-out and less space (small size). These are of two types: parallel and serial. The Charge Coupled Device (CCD) is a serial memory that needs refreshing and has long latency time. Hence it was not commercially popular. Fig. 7.7 shows the popular types of semiconductor memory.

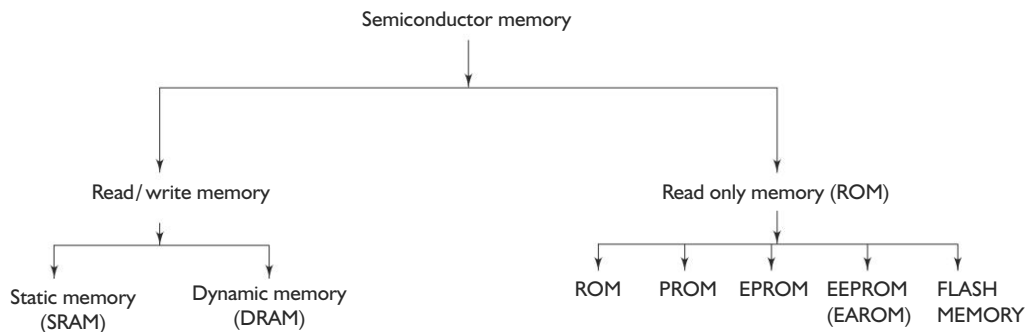


Fig. 7.7 Semiconductor memory types

7.3.5 SRAM and DRAM

Figure 7.8 illustrates a semiconductor memory cell. To perform a read or write operation, first the cell is selected by the 'select' signal. For read operation, the 'read' signal follows. The cell's content (0 or 1) is available on 'data out' (D_o) after the access time. In the case of write operation, the data (0 or 1) is sent on 'data in' (D_i) and 'write' signal is activated.

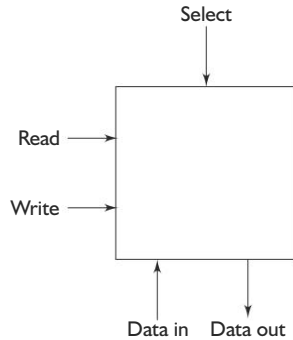


Fig. 7.8 Semiconductor memory cell

The data is stored in the cell. The semiconductor read/write memory is of two types: Static Memory (SRAM) and Dynamic Memory (DRAM). Each cell in a static memory is similar to a flip-flop. It stores a 1 or 0 as long as the power supply is present. It occupies more space as compared to dynamic memory cell but it is faster. Static memory is costly since it needs at least six transistors to build a memory cell and its packing density (amount of bits in a IC chip) is poor. It is used in applications where we need small and fast memory such as cache memory. Each dynamic memory cell is like a capacitor. Storing 1 or 0 is like storing a charge on a capacitor which gets discharged over a period of time. A dynamic memory cell can usually retain its content only for about 2 ms due to discharge. Hence before 2 ms is over, its content should be written again (recharging the capacitors). This is known as refreshing. Additional circuitry is needed to perform periodic refreshing of dynamic memory. Dynamic memory has good packing density and is cheaper and hence it is used for main memory.

7.3.6 Read Only Memory

The read only memory is classified into five types: ROM, PROM, EPROM, EAROM and Flash Memory. These are described as follows:

7.3.6.1 ROM (Read Only Memory)

The ROM is mask programmed by the manufacturer with the contents provided by the customer. The contents are fixed by metal masks during chip fabrication. Once programmed, the ROM becomes a read only memory and its contents cannot be erased. Even if a single bit is wrongly programmed, the ROM chip is useless.

7.3.6.2 PROM (Programmable Read Only Memory)

The PROM is a field programmable device. The user buys a blank PROM and enters the desired contents using a PROM programmer (burner). There are small fuses inside the PROM chip. These are burnt open (cut) during programming. The PROM chip can be programmed only once and its contents cannot be erased. It is cheaper for the customer to buy PROM from the market and program it compared to purchasing as ROM from the manufacturer.

7.3.6.3 EPROM (Erase Programmable Read Only Memory)

The user buys an empty EPROM IC from the market. A PROM programmer is used to store the contents in the EPROM. While programming, an electrical charge is trapped in an insulated gate region. To program an EPROM chip, a relatively high dc voltage of about 25 V is required. A high pulse of 10–55 ms wide is applied to the PD/PGM input. Since the EPROMs are byte organized, programming one byte at a time is done by a single pulse.

The contents of EPROM can be erased by exposing it to ultra violet light for a duration of about 30 minutes. This is done by putting the IC inside the EPROM eraser equipment, and passing ultra violet light through the quartz window (lid) on the EPROM IC. During normal use/storage of the EPROM chip, the quartz lid is sealed with a sticker. The entire EPROM is erased as a whole and selective erasing of one or more locations is not possible.

7.3.6.4 EAROM (Electrically Alterable ROM) or EEPROM (Electrically Erase Programmable ROM)

The EAROM is programmed as well as erased electrically. It can be erased and reprogrammed about 10,000 times. Any location can be selectively erased and programmed. Erasing and reprogramming is done dynamically without removing the EAROM IC from the circuit.

7.3.6.5 Flash Memory

Flash memory is a special type of EEPROM that can be erased and modified in blocks in one bulk operation (like a flash). Serial (word wise) erasing and modification is also possible. Apart from its use in a computer, the flash memory is used in several other equipments such as—mobile phones, digital camera, LAN switch, etc. In a computer, the flash memory is used in two different ways:

1. Replacement for EPROM containing BIOS. This is known as flash BIOS. This enables BIOS updates easy without removing the IC.
2. Replacement as hard disk. This is an attractive option in notebook computers and portables due to multiple advantages such as—light weight, ruggedness, low power consumption, easy portability of data.

The current trend is 'memory stick' made of flash memory that is attached to Universal Serial Bus (USB) of the personal computer so that one can carry it anywhere, like a floppy diskette, for data exchange between computers. It is popularly known as pen drive due to its shape.



7.4 ROM Subsystem Design

The ROM ICs are generally slower than RAM ICs. They are generally byte organized. Figure 7.9 shows a block diagram for a $8\text{ K} \times 8$ ROM. Figure 7.10 shows a 256 K ROM, ($32\text{ K} \times 8$), EPROM IC. Figure 7.11 shows the design of 64 KB ROM using two such ICs.

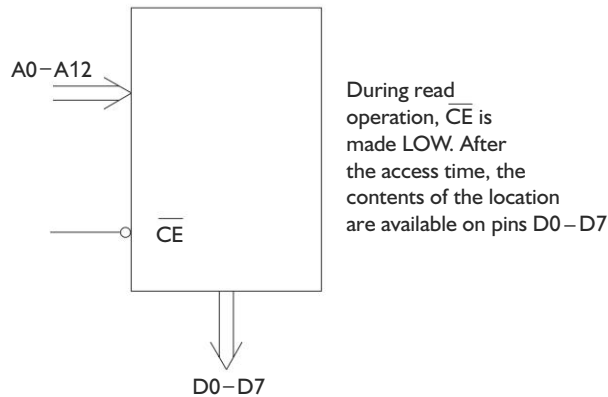


Fig. 7.9 $8\text{ K} \times 8$ ROM



7.5 Static RAM IC

In SRAM, the flip-flop is used as a one bit memory. When it is triggered by a clock signal, the synchronous data input enters the flip-flop.

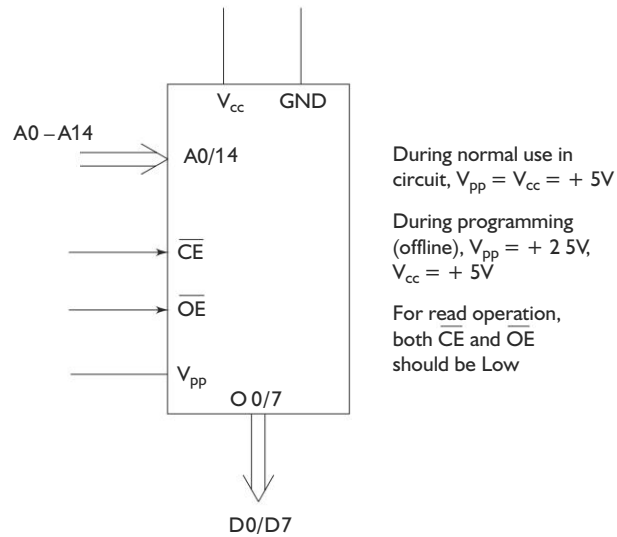


Fig. 7.10 256 K (32 K $\times$ 8) EPROM IC

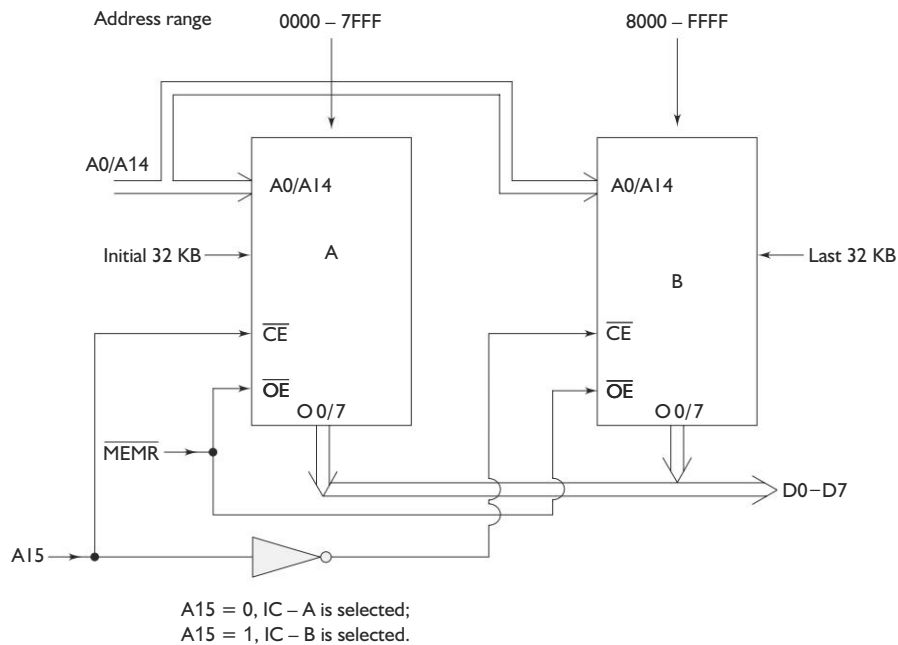


Fig. 7.11 Designing 64 KB ROM using 32 K $\times$ 8 IC

A register can be considered as a memory word. It has multiple flip-flops with a common clock input. Figure 7.12 shows a 4 bit D type register with a common clock and common reset input. A register can be seen as one dimensional memory. By combining multiple registers, a memory of any number of locations can be built. Figure 7.13 shows a two-dimensional memory using multiple registers. As the number of locations increases, the length of address decoder also increases. For example, to build a 1 KB memory, 1024 registers of 8 bits are required. A 10 to 1024 decoder is needed to decode 10 address bits, which means a large number of multiple input gates are required. Hence this approach has practical difficulty. A possible solution is arranging the memory elements (cells) as a matrix.

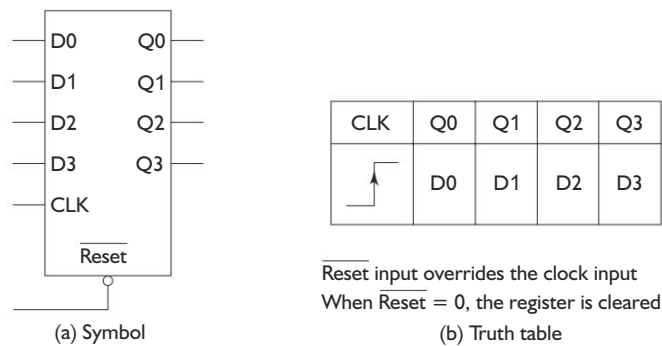


Fig. 7.12 4-bit register

Figure 7.14(a) shows a block diagram of a $1\text{ K} \times 4$ SRAM IC. It has 1024 locations of 4 bits each. Hence 10 bits are needed for addressing this memory chip. A read or write operation is possible only when the CS (chip select) input is active (LOW). The exact operation required is controlled by the Write Enable (WE). When it is low, the IC performs a write operation. It takes the data (4 bits) from the data pins and stores in the location given by the address bits. When the WE input is high, the IC performs a read operation of the location and supplies its contents on data pins. Its internal organization is shown in Fig. 7.14(b). Internally, the memory cells are organized as a 64×64 matrix. Eight bits of the address are decoded by the row address decoder to select the row. The column cells are arranged as 16 sets of each set having four cells as shown in Fig. 7.14(c). The remaining four bits in the address point to one of the sets. Hence during the read operation, a 16-to-1 MUX of 4-bits is used to select the set using the 4 bits as select control inputs to multiplexor. During write operation, the demultiplexor routes the data input to the set corresponding to the four address bits.

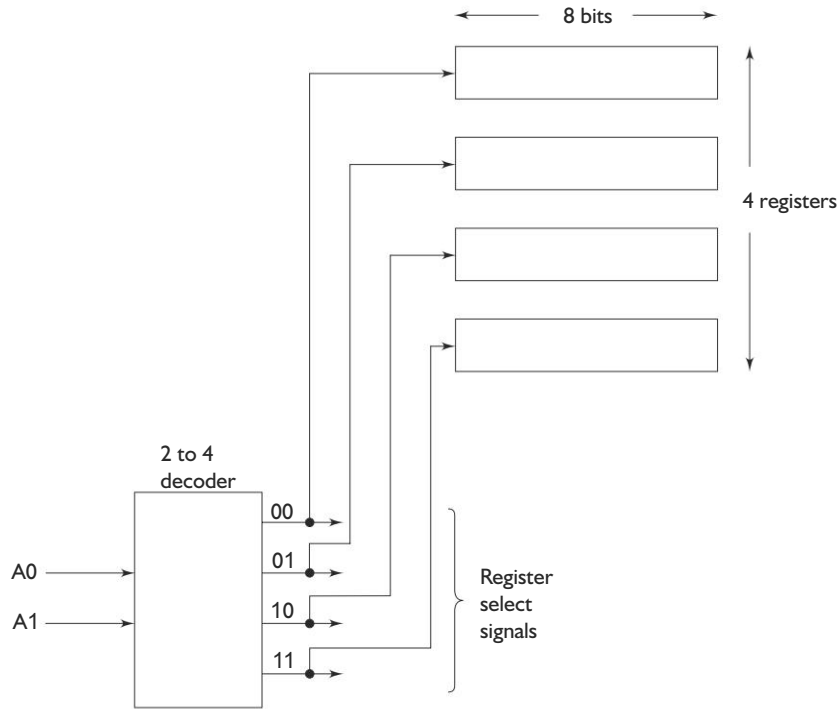


Fig. 7.13 4 x 8 memory using four registers

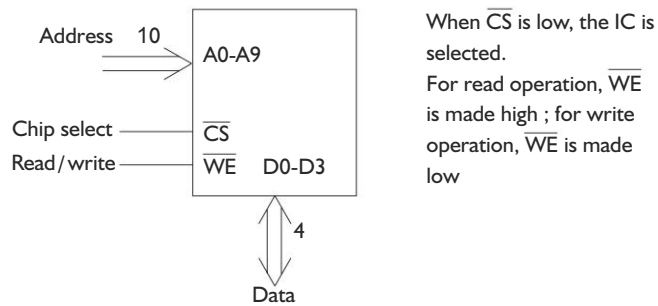


Fig. 7.14 (a) SRAM IC of 1 K x 4

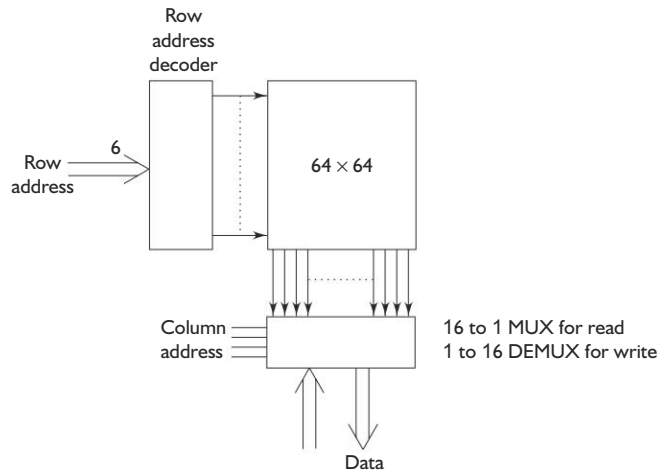


Fig. 7.14(b) Internal organization of 2114 SRAM

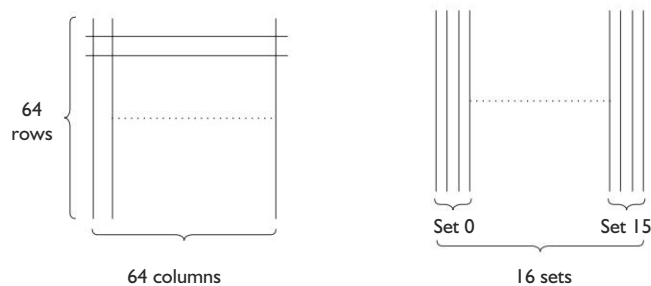


Fig. 7.14(c) Matrix organization of 2114 SRAM

7.5.1 Design of SRAM Subsystem

Using multiple SRAM ICs of same type, a large capacity SRAM memory module can be designed. Figure 7.15 shows the construction of 4 KB RAM using SRAM IC of $1\text{ K} \times 4$. Each bank has two ICs together forming 1 KB. The CS inputs of the two ICs are connected (shorted) together. The four banks together form 4 KB. The bank that should participate in a given read/write operation is selected by decoding the two msbs of the 12 bit address. The decoder output goes as CS input to a bank. The WE input of all eight ICs are connected together and controlled by READ/WRITE signal. The address pins A9–A0 of all eight chips are joined together. The ICs are divided into two groups for tying the data pins. One

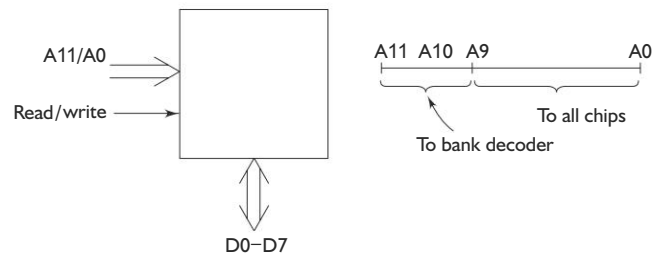


Fig. 7.15(a) Symbol for 4 KB SRAM

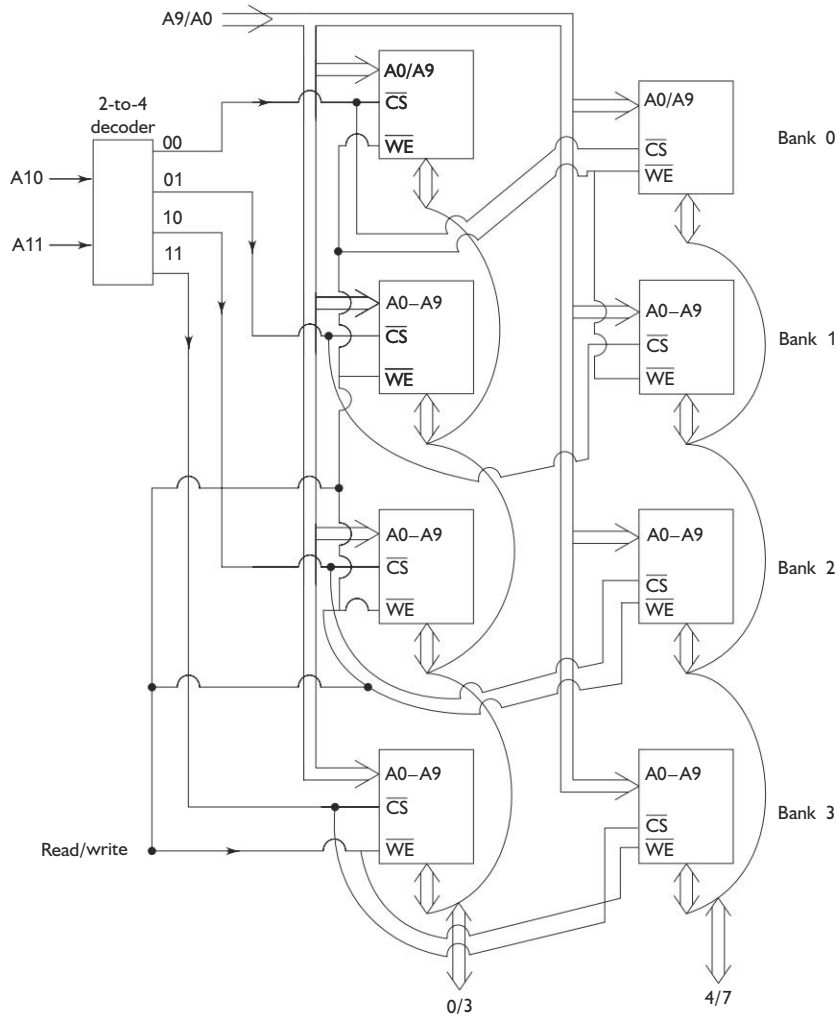


Fig. 7.15(b) Logic scheme for 4 KB SRAM

chip in each bank provides four bits (half) of the byte. The D0 of the corresponding chips in each bank are connected. Thus, the D0–D3 are formed by four chips and D4–D7 are formed by the remaining four chips. When the CPU sends a 12-bit address (A11–A10), the A11 and A10 are used to select a bank and the A9–A0 are used to select the location within the bank.

Example 7.1 The 2167 is a SRAM IC of $16\text{ K} \times 1$. How many such ICs are needed to provide 64 KB RAM?

Each IC gives 1 bit. Hence to form a byte, 8 ICs are needed. Since each IC has 16 K locations, eight ICs give 16 KB capacity. Hence, we need $4 \times 8 = 32$ ICs to obtain 64 KB RAM. These are organized as four banks. Each bank provides 16 KB.

Example 7.2 A 32 KB RAM is formed by 16 numbers of a particular type of SRAM IC. If each IC needs 14 address bits, determine:

(a) the IC capacity; (b) memory organization; (c) address range for each bank.

(a) Total capacity = 32 KB = $32\text{ K} \times 8\text{ bits} = 256\text{ K bits}$.

Since 16 ICs are used, each IC provides, $256\text{ K bits} \div 16 = 16\text{ K bits}$.

Hence, capacity of each IC is 16 K bits.

Since, 14 bit address gives 16 kilo locations, each chip has 16 K locations.

Hence, the IC organization is $16\text{ K} \times 1$

i.e. capacity of each IC is 16 K bits.

(b) The RAM module is formed by two banks. Each bank gives 16 KB RAM.

(c) To address 32 KB memory, 15 bits are needed in the address, denoted as A14 –A0.

The A14 selects the bank. When A14 = 0, bank 0 is selected and when A14 = 1, the bank 1 is selected. The address ranges are as follows:

Bank 0: 0000000000000000 to 0111111111111111; in hexa decimal, the range is 0000 to 3FFF.

Bank 1: 4000 to 7FFF

7.6 Dynamic RAM

Though DRAM is slower than SRAM, DRAM is widely used as the main memory due to its cost effectiveness. Additional circuitry is needed for the refresh cycles but it does not cost much. However, DRAM is less reliable than SRAM and therefore error detection mechanism such as parity logic is essential for DRAM. Additional memory chips are needed for storing parity bits. In high end computers, a more sophisticated mechanism such as Error Checking and Correcting code (ECC) is employed in memory modules to offer error detection and automatic correction. This improves the reliability of the memory subsystem.

7.6.1 DRAM Refresh

The content of each bit cell in the DRAM chip has to be refreshed once in every few (1, 2 or 4) milliseconds. If each location is accessed within this duration, there is no need for separate refreshing. However, the possibility of the CPU accessing each location regularly is remote. Hence we need a separate mechanism for memory refresh.

Theoretically, for refresh operation, the contents of each location should be read from the DRAM and written back. Practically, the DRAM chips have a built-in refresh capability. It is enough to give the location address and refresh signal input periodically. Reading and writing back the contents is done internally in the DRAM IC. Whenever the REFRESH input is low, the memory chip refreshes the contents of the location whose address is given in the address inputs. During refresh, the memory does not use DATA pins and these pins are tristated. Current DRAM ICs do not have a separate input pin for memory refresh. Instead, the DRAM IC is indirectly informed of the refresh operation as discussed in the next section.

7.6.2 Refreshing by Rows

Since internally the DRAM has a matrix of rows and columns, refreshing can be done row-wise. When one row is refreshed, all bit cells in that row are covered. Thus the number of refresh sequences required to cover all the bit cells once is equal to the number of rows. During each refresh, the address of the row is supplied to the DRAM ICs. In general, the refresh schemes are divided into two categories:

1. Burst refresh
2. Distributed refresh

In the *Burst refresh* scheme, all the rows are refreshed one after another in one stretch. This is repeated every 2 ms (or 4 ms). Hence during one complete cycle of burst refresh, no read/write operation takes place. In *Distributed refresh* scheme, refreshing of rows is done between read and write operations.

Example 7.3 The 4416 DRAM IC ($16\text{ K} \times 4$) has an access time of 150 ns. Its internal organization is 256 rows by 256 columns. Its memory refresh specification is 256 cycles/4 ms. Calculate the refresh overhead.

In 4 ms, 256 cycles of refresh take place.

Since access time is 150 ns, refreshing 256 rows take a time of $256 \times 150\text{ ns}$.

This is equal to 38.4 μs .

This much time is spent in refresh in every 4 ms duration.

$$\begin{aligned}\text{Hence refresh overhead} &= \text{Time taken for refreshing all rows once} \div \text{refresh interval} \\ &= (38.4 \times 10^{-6}) \div (4 \times 10^{-3}) = 9.6 \times 10^{-3} \times 100\% \\ &= 0.96\end{aligned}$$

Roughly 1% is the refresh overhead.

7.6.3 Row and Column Address Multiplexing

A technique followed by DRAM IC manufacturers, to reduce the pin count and the IC size, is multiplexing row and column address on one set of pins. Figure 7.16(a) illustrates this for the 4164 IC, a 64 K bit DRAM IC of $64\text{ K} \times 1$ organization. Fig. 7.16(b) shows the internal organization of the 4164 IC. The 4164 IC has memory cells in a 128×256 matrix. There are two such matrix cells in a single IC. Effectively, it is equivalent to a matrix of 256

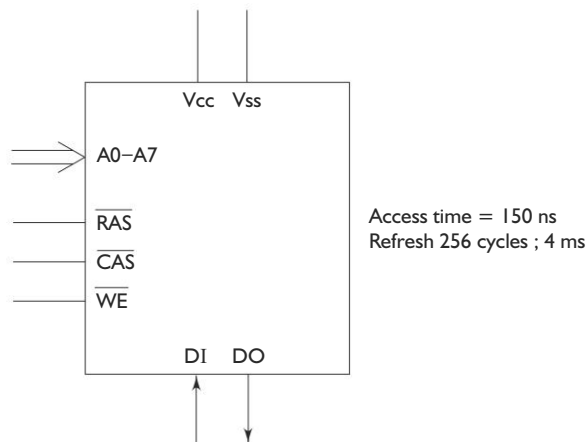


Fig. 7.16(a) Pin-out of 4164 DRAM IC

rows and 256 columns. An address to the memory chip points to a particular cell in the matrix. The memory chip has internal decoders for row address and column address. Since there are 64 K locations, 16 bits are needed for addressing. The 16 bit address contains 8 bit row address and 8 bit column address. As shown in figure, instead of 16 pins for address, only 8 pins are present. The same pins are used for giving row address first, and then column address. The $\overline{\text{RAS}}$ and $\overline{\text{CAS}}$ signals are used by the chip to latch the row address and column address, respectively. When row address is given, the $\overline{\text{RAS}}$ input is made active (low). Subsequently, when the column address is sent, the $\overline{\text{CAS}}$ input is made active. Internally, the DRAM IC has a row address latch and a column address latch. When the $\overline{\text{RAS}}$ goes from high to low, the chip latches the address into the row address latch. All the cells corresponding to this row are read. When $\overline{\text{CAS}}$ goes from high to low, the chip latches the address into the column address latch. Though all the cells in a row have been read, only the output of the cell corresponding to this column is enabled on data line. The read or write operation is indicated to

the chip by the signal $\overline{WE}$. For write operation, the $\overline{WE}$ is made low and for read, it is high. The sequence of operations for read/write process is as follows:

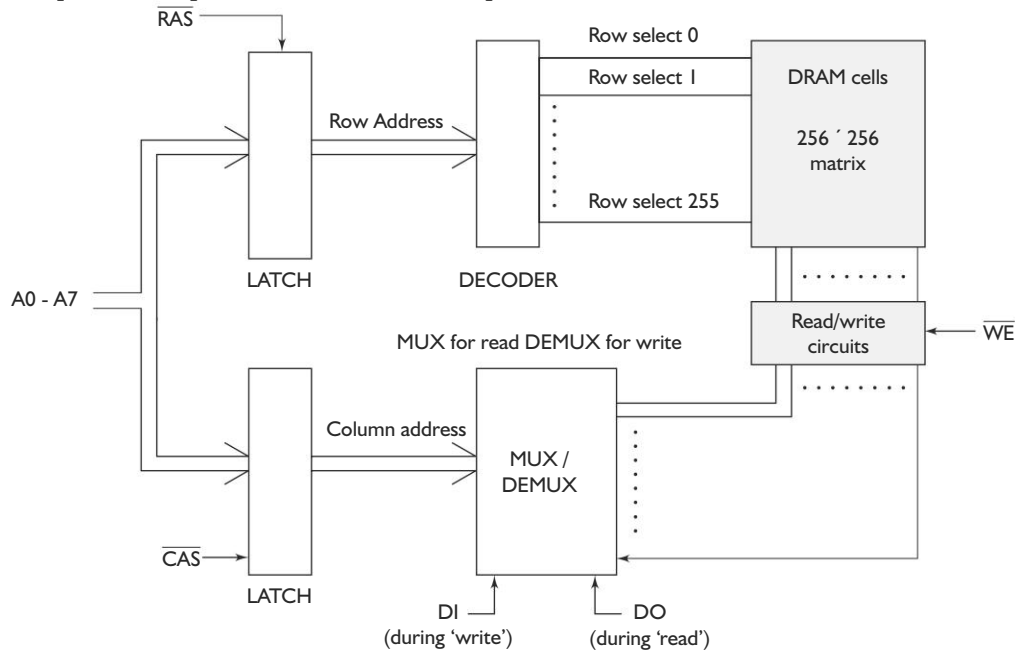


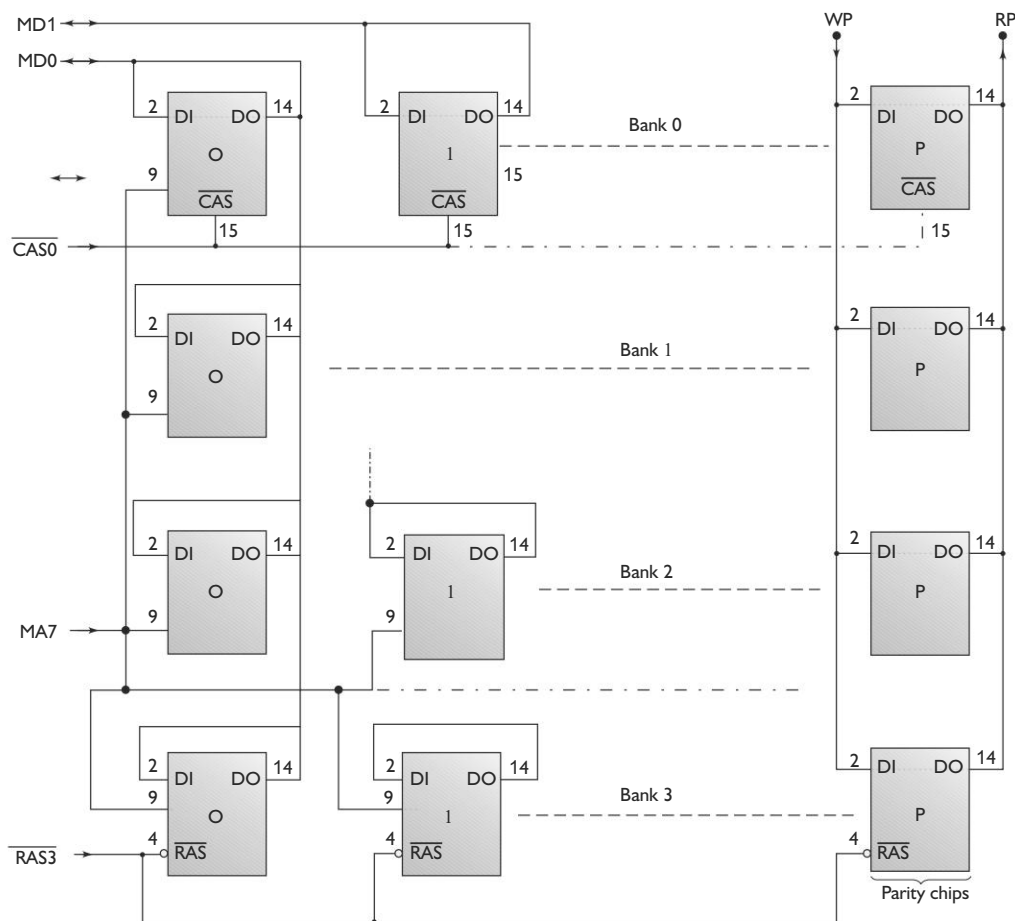
Fig. 7.16(b) Internal organization of 4164 IC

1. Supply 'row address' on address pins
2. Activate $\overline{RAS}$; 'row address' enters the row address latch.
3. Supply data bit in case of 'write' operation
4. Make $\overline{WE}$ low for 'write' and high for 'read'
5. Supply 'column address' on address pins
6. Activate $\overline{CAS}$; column address enters the column address latch.
7. Receive data bit in case of 'read' operation

7.6.4 Design of DRAM Subsystem

Suppose we need to build a memory module of 256 KB using 64 K DRAM IC. Figure 7.17 shows the block diagram of memory module (with parity bit for every byte). The 36 DRAM chips are arranged as four banks of 9 chips per bank. The address bits of all the chips are connected to the corresponding pins of other chips. So we have 8 bit address, MA0/MA7,

for the memory module. The parity chip has pin no. 2 as input and pin no 14 as output. For other chips, pins 2 and 14 are shorted externally. Hence, for each bank, the MD0/MD7 is bi-directional data. The data pins of each chip are connected to data pins of corresponding chips in the other banks. The $\overline{\text{RAS}}$ input of each chip is connected to $\overline{\text{RAS}}$ inputs of other chips in the same bank. Similarly, the $\overline{\text{CAS}}$ input of each chip is connected to $\overline{\text{CAS}}$ inputs of other chips in the same bank. Thus the banks have $\overline{\text{RAS}}0$ to $\overline{\text{RAS}}3$ and $\overline{\text{CAS}}0$ to $\overline{\text{CAS}}3$ as inputs. The $\overline{\text{WE}}$ of all the chips are connected together. With these interconnections, the DRAM appears as a black box shown in Fig. 7.18.



Note: not all the interconnections are shown

Fig. 7.17 RAM banks organization—256 KB DRAM using 64 K × 1 ICs

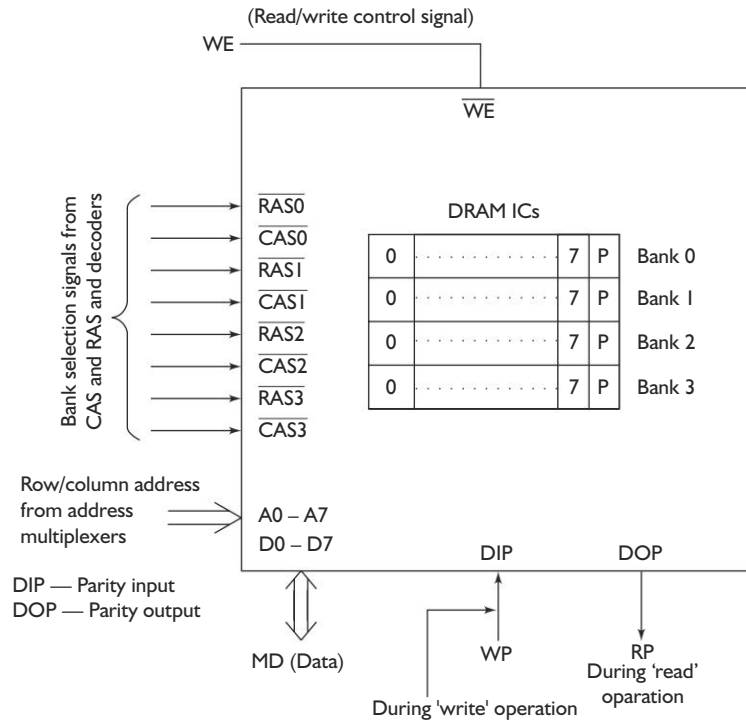


Fig. 7.18 256 KB RAM subsystem as a black box

7.6.4.1 RAM Control Signals Generation

Figure 7.19 shows a block diagram for generation of RAM control signals. Figure 7.20 shows a timing diagram of control signals and data output during read operation. The CPU first sends a 16 bit address. This is divided into two halves: the most significant half gives the column address and the least significant half gives the row address. The row address and column address are routed one after the other with some time delay on the address pins, along with corresponding control signal $\overline{\text{RAS}}$ or $\overline{\text{CAS}}$. The processor issues MEMORY READ control signal in the next clock. Immediately, the RAS signal goes high. Then the ADDR SEL signal goes high after a delay of 60 ns. This signal controls the select input of address multiplexers. During the initial 60 ns, the ADDR SEL signal is low. Hence the address multiplexers select the row address bits. After 60 ns, the ADDR SEL is high and the address multiplexers select the column address bits. The $\overline{\text{CAS}}$ signal is generated 100 ns from RAS (40 ns from ADDR SEL). Thus, there is 40 ns settling time, for the column address, to stabilize at the inputs of DRAM chips, before $\overline{\text{CAS}}$ is applied to the DRAM

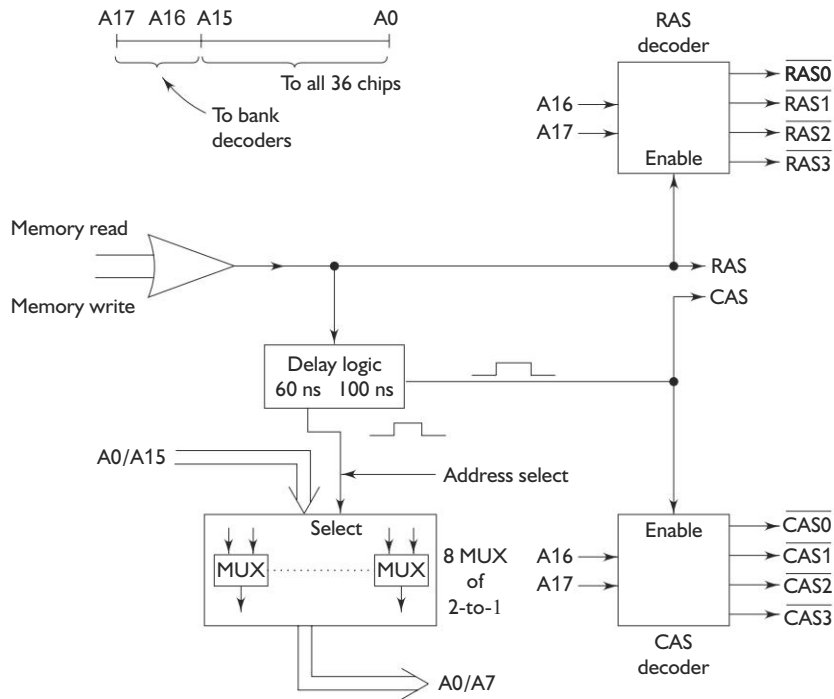


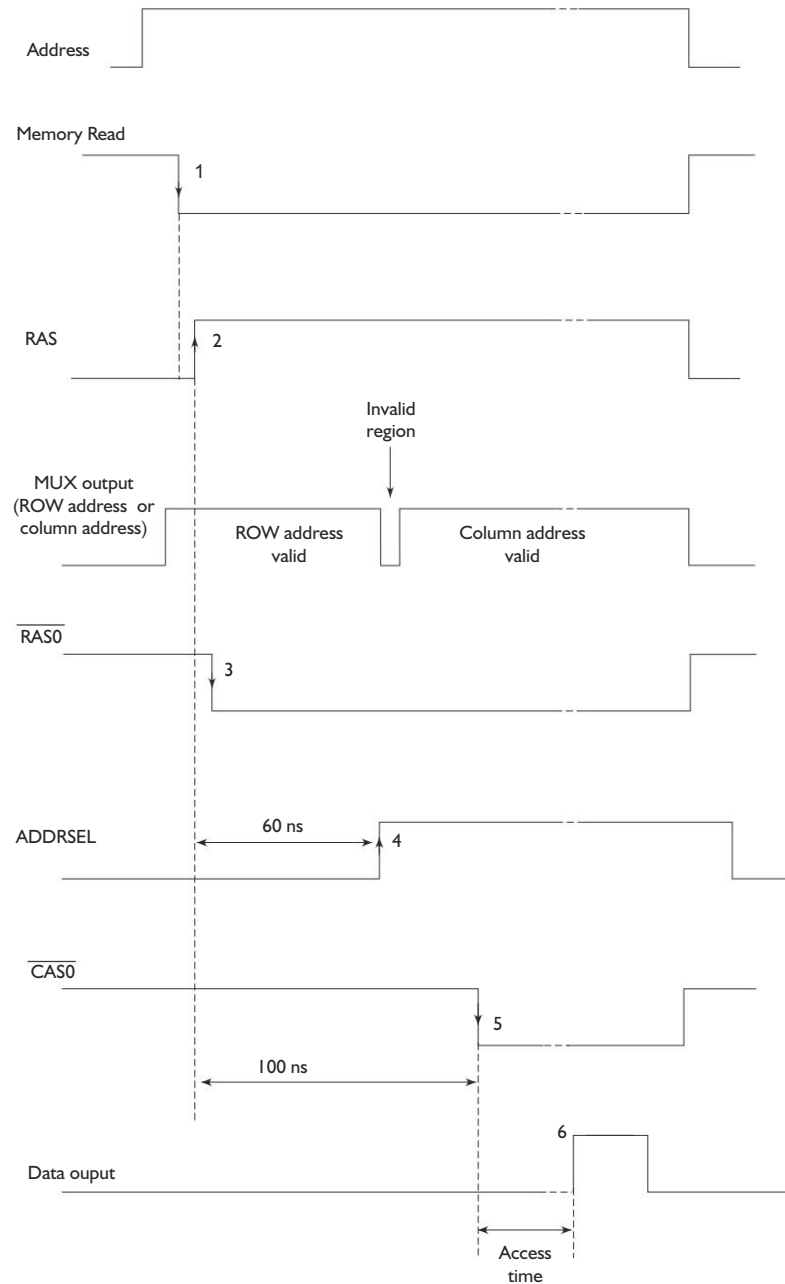
Fig. 7.19 RAM banks selection for 256 KB RAM

chips. For any 16 bit address, the corresponding bank should be enabled and others should be disabled. This is done, by the bank selection logic, by decoding the most significant address bits A16 and A17. In addition, the following functions are performed by the bank selection logic (not shown in the diagram):

1. Disable the $\overline{\text{CAS}}$ and $\overline{\text{RAS}}$ decoders during refresh cycles.
2. Activate $\overline{\text{RAS}}$ signals of all four banks during refresh cycle. Due to this, all the 36 chips perform refresh of a particular row, simultaneously.

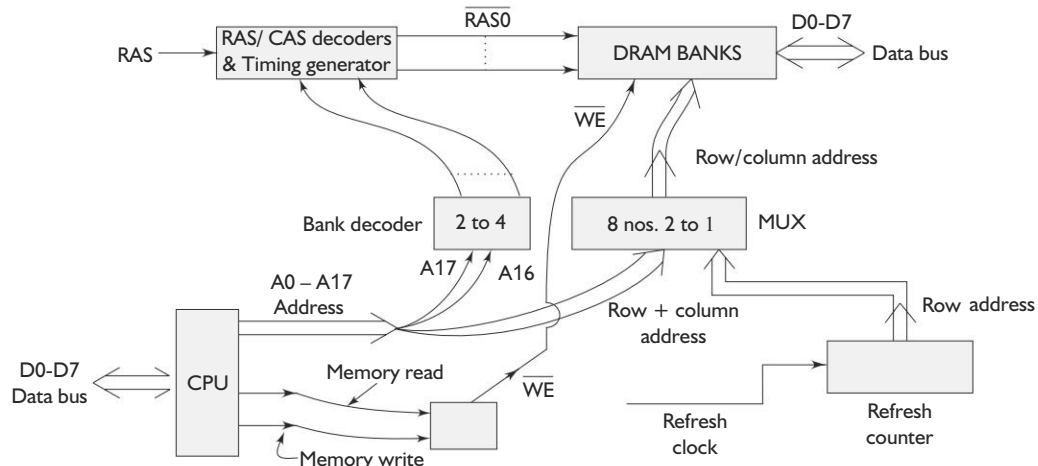
7.6.4.2. Refresh Mechanism

The address of the row to be refreshed is maintained by a counter that increments the contents, after every refresh cycle. Also a refresh clock is required to indicate that it is time for a refresh cycle. There are several ways for doing this. In IBM PC, one channel of DMA controller (for row address) and one timer (for refresh clock) are dedicated for this purpose. In many small systems, a simpler logic is followed with a dedicated refresh counter. See Figure 7.21.



Note: The numbers 1 to 6 indicate the timing sequence for important signals.

Fig. 7.20 RAM control signals—timing diagram for read operation



Note: Not all the control signals are shown

Fig. 7.21 DRAM module with refresh logic

Example 7.4 A computer uses 41256, a DRAM IC of $256\text{ K} \times 1$. Externally, it appears to be a matrix of 512 rows and 512 columns. Show the RAM banks organization for a 1 MB RAM module using these chips. Assume no parity bit requirement.

We need four banks of 8 chips in each bank to form a 1 MB memory. We need 20 bit address A0–A19. The two msbs A19 and A18 are used for selecting the bank. The remaining address bits A17–A0 are split into row address bits A8–A0 and column address bits A17–A9. Modify Figures 7.17–7.19 suitably.

7.6.4.3 Pseudo Static RAM

A DRAM that does not need refresh cycles but requires periodic timing information is known as pseudo static RAM. There is no need for conducting the refresh cycles by external hardware. The IC internally maintains the row address to be refreshed.

7.6.4.4 Cache DRAM (CDRAM)

The CDRAM is a DRAM chip with an on-chip cache memory.

7.6.4.5 Fast Page Mode (FPM)

In FPM technique, the DRAM first receives an address for a read operation. The subsequent addresses are carefully chosen so that they have the same row address as the first address. Thus, the DRAM does not require the row address for the subsequent reads.

Internally, the DRAM need not select the row again. The contents of the row of the first read operation are used for subsequent read operations. It can supply contents corresponding to various columns from internal buffer. Hence a block of data (called a 'page') is transferred at high speed. Thus, compared to accessing n number of random addresses, accessing n number of addresses of same row will be quick.

7.6.4.6 Asynchronous and Synchronous DRAM

The DRAM discussed so far is of asynchronous DRAM type. Asynchronous DRAM does not maintain timing information. The DRAM control logic in CPU knows the access time of the DRAM and generates the RAS and CAS signals at appropriate time. In case, the access time of DRAM is more compared to CPU's clock, the CPU is forced to enter wait state for one or more clocks as required. Hence, use of asynchronous DRAM requires timing control and wait states during CPU's memory read or write sequence. Figure 7.22 shows the interface for an asynchronous DRAM.

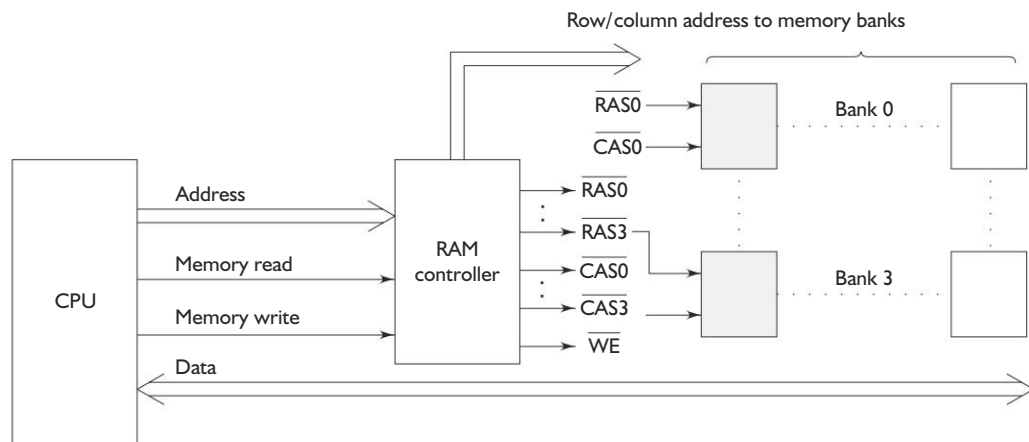


Fig. 7.22 Asynchronous DRAM interface

A synchronous DRAM gets clock signal also along with other signals from the CPU. Figure 7.23 shows the interface between the CPU and SDRAM. The SDRAM is more intelligent than asynchronous DRAM. It has multiple modes of operation. The exact mode for any read/write operation is fixed by the contents of the mode register in the SDRAM. The CPU selects the required mode by writing the control information in the mode register. The mode register enables programming of SDRAM to match the specific speed requirements of CPU. The burst length and latency are some of the programmable parameters.

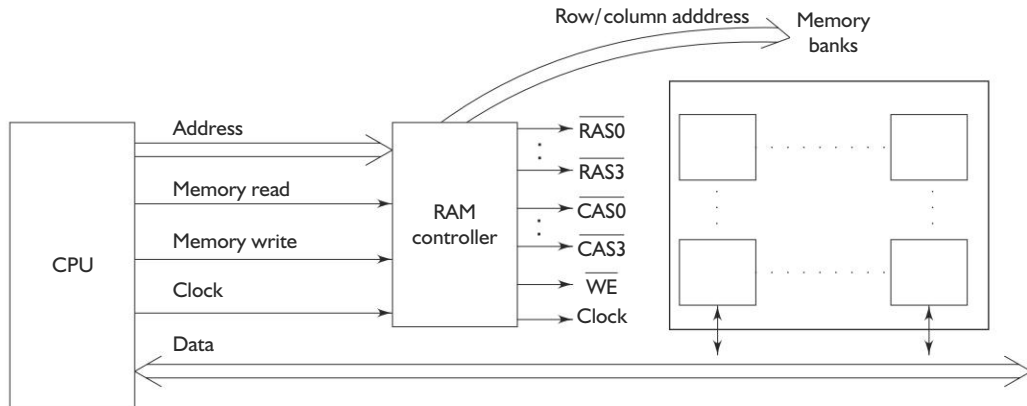


Fig. 7.23 Synchronous DRAM interface

The CPU provides the address, the mode information and other controls. The SDRAM latches them internally. When the SDRAM performs read or write operation, the CPU does not wait for it; it performs other activities. The SDRAM synchronizes itself with the timing of the CPU. Thus, the memory controller knows when the operation is completed by the SDRAM. The SDRAM has its own refresh mechanism with a refresh counter for maintaining the row address. SDRAM also uses interleaving and burst modes to achieve faster access. SDRAM uses 'pipeline' burst mode to start a new access before the current access is complete. It provides a 5-1-1-1 pattern; the first access (latency) needs 5 clocks whereas the next 3 accesses need 1 clock each. The *latency* indicates the time taken to transfer the first byte of data. Interleaving is a technique of placing consecutive locations in different memory blocks. Figure 7.24 shows a two-way interleaving with two blocks (odd and even banks) of memory. Hence two simultaneous memory access can be performed. This doubles the effective memory bandwidth. Further discussion on memory interleaving is given in Chapter 8. The SDRAM has on-chip interleaving. Figure 7.25 gives a block diagram of SDRAM. The commercially available SDRAMs are operated at 100 and 133 MHz clock rates.

SDRAM is commercially available as standard modules incorporating multiple SDRAM ICs to form the required capacity. The DIMM SDRAM is available in 64 bit (no parity) or 72 bit (with 8 parity bits) modules in three standard clock rates: 66, 100 and 133 MHz. Some of the commercial standards of SDRAM DIMM are: 8, 16, 32, 64, 128, 256, 512 and 1024 MB. The 8 MB module uses 72 pieces of SDRAM ICs of each 1 M bits capacity. The 1024 MB module uses 64 (or 72) pieces of SDRAM ICs of 128 M bits capacity.

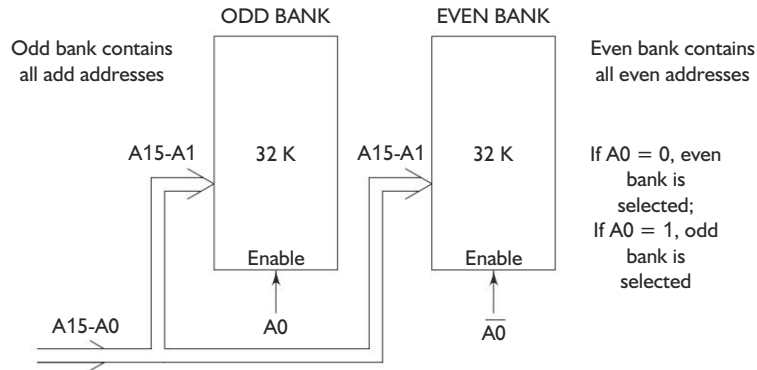


Fig. 7.24 Two-way memory interleaving

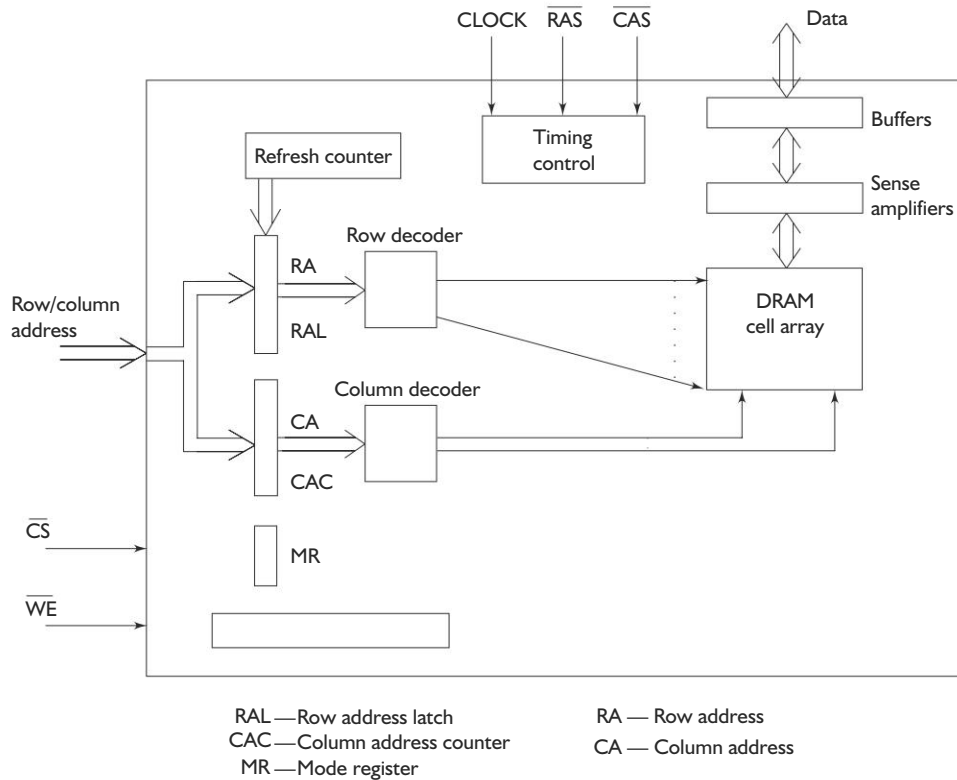


Fig. 7.25 SDRAM internal organization

7.6.4.7 Double Data Rate Synchronous DRAM (DDR SRAM)

The standard SDRAM performs its operations on the rising edge of the clock signal. But the DDR SDRAM transfers data at both edges of the clock. This doubles the data transfer rate. For example, a 100 MHz memory bus clock produces an effective data rate of 200 MHz with DDR SDRAM.

7.6.4.8 SPD and PPD

The Serial Presence Detect (SPD) and Parallel Presence Detect (PPD) are two different methods used for indicating the memory configuration to the system software. The PPD uses a number of resistors for this purpose. This is a traditional method used by SIMMs and DIMMs. The SPD uses an EPROM or flash ROM which stores information about the RAM module.

7.6.4.9 Rambus RDRAM

The Rambus DRAM (RDRAM) follows an intelligent interface protocol with the CPU. There are no address lines from the CPU to RDRAM. Instead, the RDRAM has data lines through which intelligent 'packets' are transferred between the CPU and RDRAM. These packets consist of several bytes of information. Since the data lines are only nine (including parity bit), multiple cycles are required to transfer one packet. There are three kinds of packets:

1. 'Request' packet from the master (CPU or memory controller)
2. 'Acknowledge' packet from RDRAM (slave)
3. 'Data' packet

To initiate a read or write operation, the master transmits a 'request packet' containing following information:

1. Operation type
2. Memory address (starting)
3. Number of bytes

On receiving a request packet, the slave which contains the memory address transfers the 'acknowledge packet' indicating whether it is ready or busy. In case the slave is busy, the master should try again after some time. Each RDRAM has a configuration register which defines the address range of that chip. The master initially loads proper values in the configuration registers of all the RDRAMs. This is performed by the master using the SIN (input to RDRAM) and SOUT (output from RDRAM) signals which form a daisy chain of

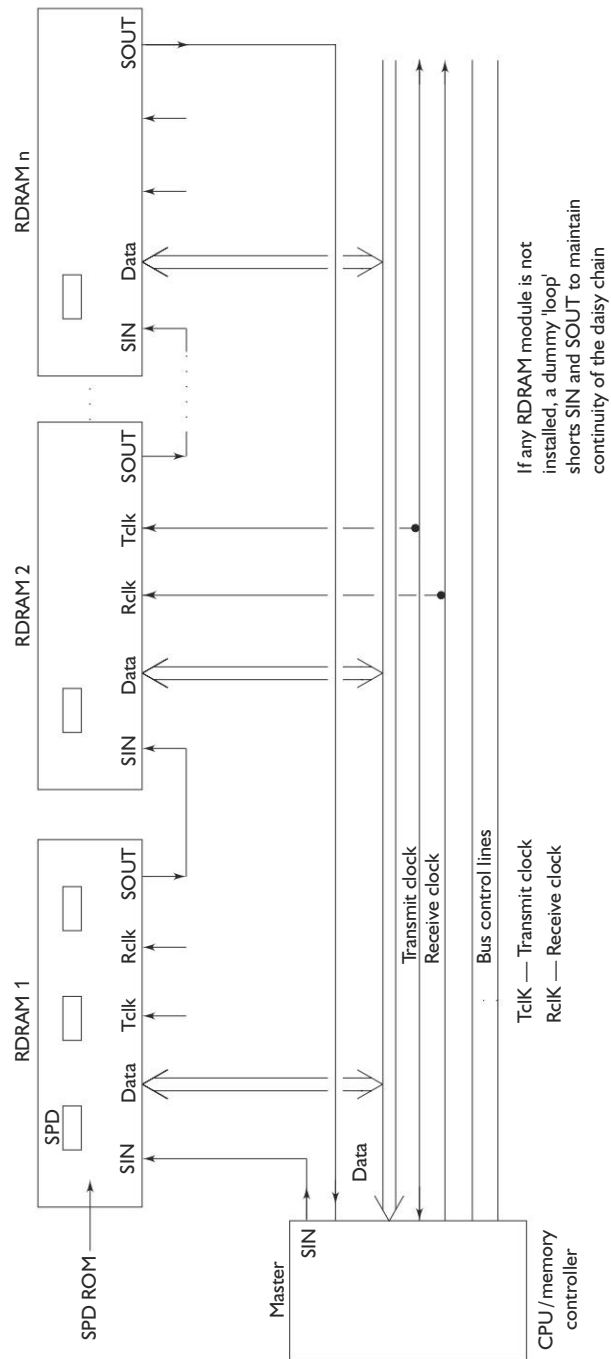


Fig. 7.26 Rambus channel

all the RDRAMs as shown in Fig. 7.26. The master performs initialization sequence for the slaves using the SIN and SOUT signals. The RDRAM chips are commercially combined together in modular form as RIMM. The RDRAM chips accept a clock frequency of 400 MHz. Its 'double clocked' feature allows operations to occur on both rising and falling edges of the clock.

The initial Rambus specification (by Rambus Inc.) listed nine data lines (including one parity bit). The two-channel Rambus has 18 data lines (two data bytes plus two parity bits). Hence, the effective data transfer rate is $2 \times 800 \text{ MHz} = 1600 \text{ MHz} = 1.6 \text{ GHz}$. This produces a bandwidth of 1.6 GB/sec.

The RDRAM module is known as RIMM. It incorporates SPD using a flash ROM. It contains information about the RIMM's size, the type and the timing information to be supplied to the master.



7.7 Main Memory Allocation

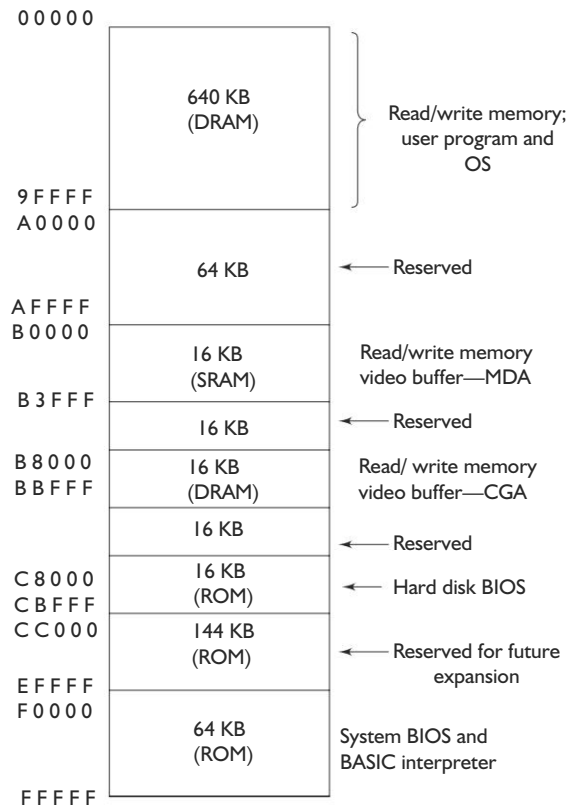
When the architecture of a computer is finalized, one of the most important design tasks is the utilization of the main memory space. Entire main memory space cannot be dedicated for program usage due to following reasons:

1. Some memory space is needed for the allotment of addresses for I/O ports in a memory mapped I/O scheme. Though number of I/O ports in a computer is in the order of 10–100, a large space of 64 K is reserved for I/O ports. This portion cannot be used for programs. The hardware is responsible for decoding this address space and disabling the main memory during read/write operation for any address in this space.
2. In some systems, a portion of main memory is reserved as video buffer (also known as screen memory). Physically, a read/write memory is present in this address space. It is a shared memory between the CPU and display (CRT) controller. The CPU stores 'pages of memory' to be displayed in the video buffer and the CRT controller displays it on CRT screen. Hence this portion is not available for usual program.

Figure 7.27 shows how the main memory space of 1 MB in an IBM PC, based on INTEL 8088, is allotted to perform various functions. This allocation is known as memory map and is an important specification that is followed both by the hardware designers and software developers. Since IBM PC follows I/O mapped I/O instead of memory mapped

I/O, there is no reservation for I/O ports. The available program memory is used in two ways:

1. 640 KB read/write memory is for user program and OS.
2. The ROM is mainly for the BIOS.



Note: This figure shows the memory map for initial IBM PC based on Intel 8088. There are several changes for later PCs.

Fig. 7.27 Memory map for PC

The BIOS also includes self-test routines and ‘boot strap loader’ apart from I/O drivers. The original IBM PC also had BASIC language interpreter in ROM. Figure 7.28 shows the design of a complete memory system for a microcomputer. This design adopts the memory map shown in Fig. 7.27 for IBM PC. While calculating the overall access time for read or write, one should take into account the access time of memory chip module and the delay introduced by various address decoders and data buffers.

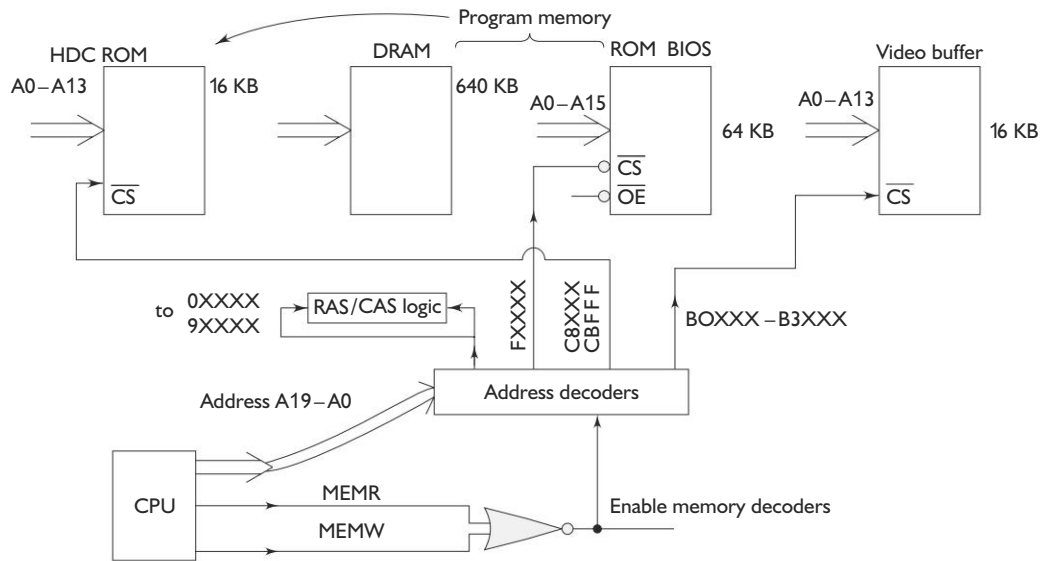
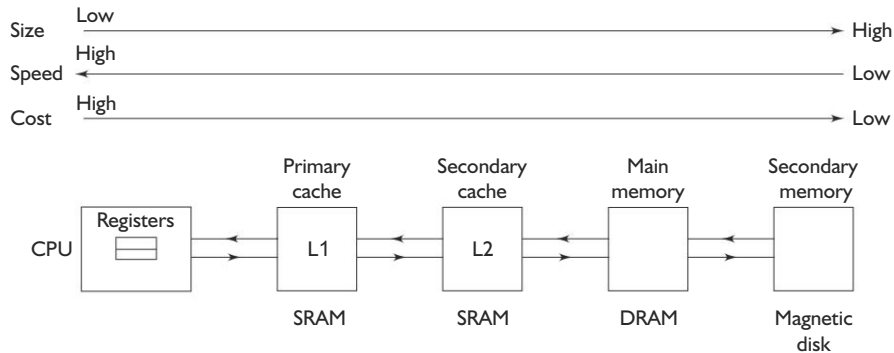


Fig. 7.28 Memory types and decoding in microcomputers

7.8 Memory Hierarchy and Performance

Ideally, the main memory should be fast and large so that a large program can be quickly stored and executed. There are different memory technologies such as semiconductor memory, magnetic hard disk, magnetic tape, optical disk etc. Each has a different range of access time. Their cost per bit also varies. Not all the memory technologies can be used for main memory. A random access memory is the right candidate for main memory (program memory). A serial access memory (magnetic tape) or semi random access memory (magnetic hard disk) does not fit as main memory even if it offers a high bandwidth. It serves best as an auxiliary memory or secondary storage to store a large volume of programs. Though slow, it is so cheap that a large amount of secondary storage can be used inside a computer system. A judicious mix of main memory capacity with secondary storage offers optimum performance in a computer system. The hardware and operating system cooperate in moving the required parts from the secondary memory to main memory at relevant time so that the CPU is not held up for want of instruction or data while making an access to the main memory. A practical computer, uses a memory hierarchy as shown in Fig. 7.29.

**Fig. 7.29** Memory hierarchy

The memory item closest to the CPU in the figure has highest speed (lowest access time) and highest cost per bit. Hence its capacity is smallest in the computer. While moving away from the CPU i.e. from registers to secondary memory, the speed as well as cost reduces from one level to next level. Hence, the capacity is increased from level to level. We try to maximize the capacity while moving away from the CPU. There are three objectives:

1. Total memory capacity should be as large as necessary.
2. The total cost should be as small as possible.
3. The average waiting time of the CPU should be as small as possible.

Nowadays, the popular secondary memory used is magnetic hard disk. A single hard disk drive offers a large storage capacity of 500 GB and more. A computer can have multiple hard disk drives. In earlier times, computers used magnetic drum which has become obsolete by the invention of magnetic hard disk. The hard disk is faster than magnetic drum by several times. The relative speed parameters are covered in Chapter 9. The optical disk technology offers better reliability as compared to magnetic hard disk but it is slow. Hence, it cannot replace the hard disk. Instead it is used as a back-up storage as a stand by for the hard disk to ensure recovery in case of loss of data.

Now a days, the DRAM is used as the main memory in addition to the small amount of ROM for storing permanent control programs. The SRAM is faster than DRAM but cost per bit of SRAM is higher. Thus, large amount of SRAM is not affordable by most computers except the high performance computers such as super computers which can have a large amount of SRAM.

SRAM is chosen for cache memory due to its high speed and the size of cache memory is also small. The cache memory can be placed in more than one level. Most of the recent microprocessors—starting from Intel 80486—have ‘on-chip cache memory’ also known as internal cache. High performance microprocessors such as Pentium pro and later have two or more levels of cache memory on-chip. These are known as level 1 (L1) level 2 (L2) etc.

caches. An on-chip cache is slightly faster than an off-chip cache of same technology. This is because of the nearly zero propagation delay in the path between the processor and the on-chip cache.

The registers inside the processor are accessed by the CPU instantly. Hence it is tempting to use as many registers as possible but the registers are expensive and the processor chip size increases heavily due to large number of registers.

SUMMARY

Main memory is the memory from where the CPU fetches instructions during program execution. For executing a program, it has to be transferred to main memory. It is also known as program memory or primary memory. A Random Access Memory (RAM) allows access to any location without any relation to its physical position and is independent of other locations. In other words, access time is equal for all locations. The read/write memory allows both read and write operations. Magnetic core memory was widely used as the main memory for several years in mainframes and minicomputers. Today, it has been totally replaced by semiconductor memory. The merits of semiconductor memory over core memory are its low power consumption, low access time, non-destructive readout and less space (small size). The semiconductor read/write memory is of two types: Static Memory (SRAM) and Dynamic Memory (DRAM). Each cell in a static memory is similar to a flip-flop. In DRAM, each cell is like a capacitor. It has to be refreshed periodically. Otherwise its contents will be lost.

The present day DRAM chips have built-in refresh capability. With these chips, there is no need to read or write back data. Instead, it is enough to provide the location address and refresh signal input periodically. Reading and writing back the contents is carried out internally by the DRAM IC. When one row is refreshed, all bit cells in that row are covered. Thus, the number of refresh cycles required to cover all the bits cells *ONCE* is equal to the number of rows. During each cycle, the address of the row has to be supplied to the DRAM ICs.

The RAM is used to store the operating system and user programs. The Read Only Memory (ROM) stores permanent control programs. It is classified into five types: ROM, PROM, EPROM, EAROM and Flash Memory.

The memory ICs are available in different capacities. The available main memory address space in a computer is distributed among RAM and ROM. The RAM and ROM circuits are designed by cascading several memory ICs. Decoding of address space and selecting relevant memory modules is performed by memory control logic.

A computer uses a memory hierarchy. A judicious mix of small main memory capacity with large secondary storage offers optimum performance. In addition, a few registers and a very small cache memory add to performance at low cost. These two techniques make it appear as if we have fast and large main memory.

REVIEW QUESTIONS

1. A designer wants to use hard disk as main memory instead of semiconductor memory. This idea can be easily rejected since the hard disk is a sequential access memory. However, if we accept a large access time, is it possible to build a RAM around the hard disk, with necessary hardware and/or software, so that it can be used at least for experimental purposes? How?
2. The SRAM is more reliable than DRAM but costlier than DRAM. One designer wanted to use SRAM for the OS area and DRAM for the user program area. Identify the problems that will be caused if this idea is accepted.
3. The flash memory is widely used instead of hard disk, in portable PCs such as notebooks, in order to reduce weight and increase ruggedness. Why is it not applicable in desktop computers and server systems?

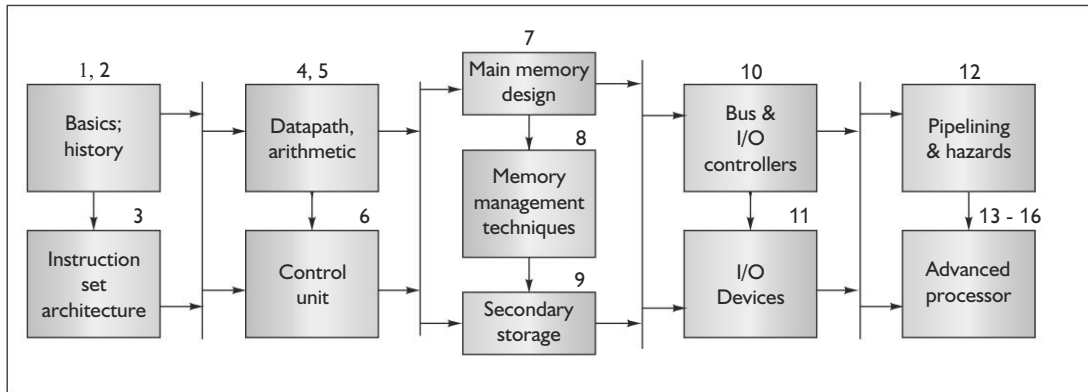
EXERCISES

1. The 85285 is a $4\text{ M} \times 9$ SIMM (Single Inline Memory Module) that costs Rs. 100. Calculate the cost of 64 M words of 32 bits each for the computer. The computer has one parity bit for every byte.
2. The 4164 is a 64 K DRAM ($64\text{ K} \times 1$) whose access time is 100 ns. Its internal organization has 256×256 matrix (internally it has two 32 K bit array). If refreshing is done with RAS-only principle, calculate the refresh overhead.
Note: Each cell must be refreshed atleast once in 2 ms.
3. The 814400 is a DRAM IC of $1\text{ M} \times 4$. Its parameters are as follows:
 - (a) Cycle time: 80 ns
 - (b) Refresh time: 60 ns
 - (c) Refresh cycle: 1024 cycles; 16.4 msCalculate the refresh overhead.
4. A computer's memory is built using 81C4256 IC that is a $256\text{ K} \times 4$ DRAM. Calculate the number of chips needed to get a memory capacity of 4 MB.

CHAPTER 8



MEMORY MANAGEMENT TECHNIQUES



Chapter Flow Diagram

OBJECTIVES

- « *Instruction prefetch*
- « *Memory interleaving*
- « *Write buffer*

- « *Cache memory*
- « *Virtual memory*

KEYWORDS

Instruction Queue	Page Fault
Bank Selection	Segmentation
Cache Line	Paging
Primary Cache	Page Table
Secondary Cache	Replacement Policy
Tag	Least Recently Used (LRU) Algorithm
Write-Through Policy	Age Register
Write-Back Policy	Address Mapping
HIT Rate	Direct Mapping
Invalid Flag	Associative Mapping
Modified Flag	Set Associative Mapping
Dirty Flag	Swapping
Cache Flush	Translation Lookaside Buffer (TLB)
Cache Invalidation	Associative Memory
Snooping	Content Address Memory (CAM)

 8.1 Introduction

The speed and capacity of the main memory impact the performance and cost of a computer system. Hence, a computer architect desires to achieve two conflicting requirements: a fast and a large memory at low cost. This chapter examines the principles and design techniques for the main memory management, to deliver better performance at low cost per bit. Some of these techniques are—memory interleaving, instruction pre-fetch, write buffer, cache memory and virtual memory. The cache memory is a system performance improvement feature. Virtual memory concept facilitates the execution of large programs without increasing the physical memory. Though cache memory and virtual memory are different concepts with different objectives, some of their implementation methods are similar. The cache memory is a hardware feature whereas the virtual memory is a hardware–software feature.

 8.2 Main Memory Drawbacks

There are two issues related to the main memory's role in a computer: speed and capacity. The slow speed of the main memory burdens the CPU. The physical capacity of main memory is important due to cost effects. Designers have found solutions to these two bottlenecks of main memory by adopting architectural changes. These are discussed below.

8.2.1 Main Memory Speed

The system performance depends mainly on instruction cycle time. The processor clock speed and the main memory access time are two major factors which contribute to the instruction cycle time. At least one main memory access is needed for every instruction—for fetching the instruction from main memory. In addition, main memory access is required for operand fetch and result storage. The processor time is wasted if memory access time is large. The following four techniques resolve this problem and enables the processor to work well with a slow main memory:

1. Instruction prefetch
2. Memory interleaving
3. Write buffer
4. Cache memory

These techniques differ in hardware complexity and efficiency (extent of reducing CPU's waiting time). Present day microprocessors incorporate all these techniques. These are purely hardware techniques invisible to the program. These are discussed in details in Sections 8.3–8.6.

8.2.2 Main Memory Size

A large main memory enables execution of large programs. Modern processors have large physical memory space. Hence, theoretically it is possible to have as much physical memory as required by the programs. But there are some practical issues:

1. Main memory cost increases and hence system cost increases.
2. Power supply requirement increases due to which the system cost increases.
3. Due to increased hardware (memory and power supply), heat generation increases requiring additional cooling.

From these observations, it is obvious that increase in the main memory size beyond a limit causes cumulative cost increase. Earlier to invention of virtual memory, the *program folding* (overlay) technique was followed to execute a large program. In this technique, a large program is divided into multiple parts by the programmer. Each part is loaded into the main memory before execution and stored on the hard disk after execution. The virtual memory feature is the perfect solution to execute long programs with limited main memory. But it imposes two penalties:

1. The programs should use logical addresses instead of physical addresses for instructions and data.
2. The execution time of program increases due to address translation (logical to physical) by the processor.

These are compensated by increased CPU speed and memory bandwidth. A detailed discussion on virtual memory is presented in Section 8.7.



8.3 Instruction Prefetch

The objective of instruction prefetch is to bring the subsequent instructions from main memory, in advance, when the current instruction is being executed by the processor. This is achieved by a set of 'prefetch buffers' in which instructions, that are fetched in advance, are stored. Figure 8.1 shows the concept of instruction prefetch. The prefetch buffers maintain an instruction queue of several instructions fetched in advance. When the processor

completes an instruction execution, it takes the next instruction from the instruction queue. Hence, the processor does not spend any time for instruction fetch. This has been made possible by the overlap of instruction fetch and execution phases. To manage this, the control unit has to be more intelligent in tracking queue status and instruction prefetches.

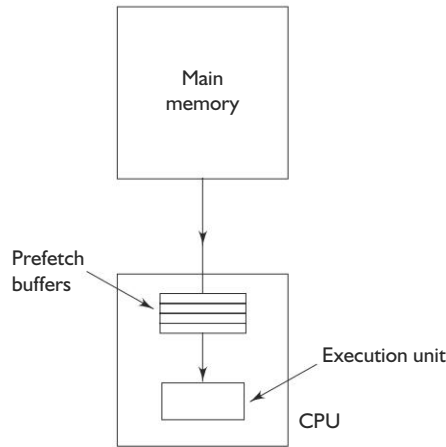


Fig. 8.1 Prefetch buffers

The instruction prefetch is a hardware feature and the program does not know about the presence of instruction queue. The control unit has to tackle a special situation also. Suppose the current instruction being executed is a JUMP instruction; then the next instruction required may not be present in the instruction queue. Hence, the queue is emptied and a fresh instruction is fetched from the jump address. As a result, the processor has to wait for one instruction fetch time after every JUMP instruction.

Example 8.1 A CPU needs 50 ns for instruction fetch. Calculate performance improvement if instruction prefetch feature is included in the processor. Assume that 20% of the instructions are branch instructions.

Case 1: Instruction prefetch feature not present

$$\begin{aligned}\text{Instruction cycle time} &= \text{fetch time} + \text{execution time} \\ &= 50 \text{ ns} + t_e \text{ where } t_e \text{ is instruction execution time.}\end{aligned}$$

Suppose a program has 100 instructions, total execution time

$$= 100 \times (t_e + 50) \text{ ns} = (100 t_e + 5000) \text{ ns}$$

Case 2: Instruction prefetch present

In 100 instructions, 20 numbers are branch instructions and others are ordinary instructions. During the execution of the program, instruction fetch by processor is needed only 20 times i.e after every JUMP instruction. For remaining 80 instructions, processor has no fetch time (due to instruction queue) and has only execution times.

Total execution time = Execution time for 80 non-jump instructions
+ Instruction cycle time for 20 jump instructions

$$= 80t_e + 20 \times (t_e + 50) \text{ ns}$$

$$= 100t_e + 1000 \text{ ns}$$

Difference in total execution time = 4000 ns over 100 instructions

Average gain per instruction due to instruction prefetch = 40 ns



8.4 Memory Interleaving

A given memory module has a fixed memory bandwidth i.e. the number of bytes transferred per sec. But the CPU is capable of handling a much higher bandwidth than the memory since the CPU usually operates at a high speed clock. Due to mismatch between memory bandwidth and CPU speed, the CPU has to idle frequently waiting for an instruction or data from the memory. Memory interleaving is a technique of reorganizing (dividing) the main memory into multiple independent modules so that the overall bandwidth is increased.

Figure 8.2 (a) and (b) illustrate a two-way interleaving for a 8 bit CPU. The CPU has 10 bit memory address. The physical memory space is 1024 bytes (1 KB). Instead of having a single memory module of 1024×8 bits, there are two modules each with 512 locations of 8 bits. Each memory module is independent and has its own address decoding and read/write control logic. Since the capacity of each module is 512 locations, 9 bits are needed to address each module. The memory address from the CPU has 10 bits, A₉–A₀. Out of these, the 9 msbs, A₉–A₁, are given as address to both modules. The lsb, A₀, controls the enabling/disabling of the memory modules. When A₀ = 0, (i.e. the address is an even address), the even bank is enabled and the odd bank is disabled. When A₁ = 1, (i.e. the address is an odd address), the odd bank is enabled and the even bank is disabled. Hence when the CPU sends a 10 bit address, only one of the banks is enabled. As far as the CPU is concerned, it has a single memory unit of 1024 locations. Now the CPU can initiate two memory reads for two different addresses, one odd and one even, one after another. Let us assume that the CPU sends address 1000 first and address 1001 next. When address 1000 is received, even

bank is selected and it starts read operation. When address 1001 is received, the odd bank is selected, and it also starts read operation. Thus, the banks are selected one-by-one and both banks perform read operation in parallel. The CPU receives the data from the two banks in the same order.

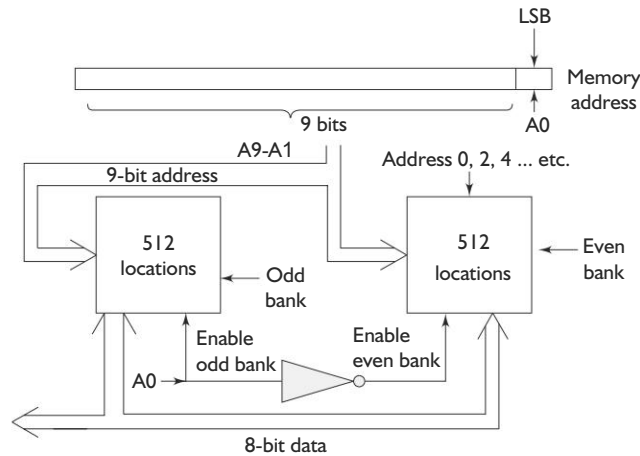


Fig. 8.2(a) Two-way interleaving

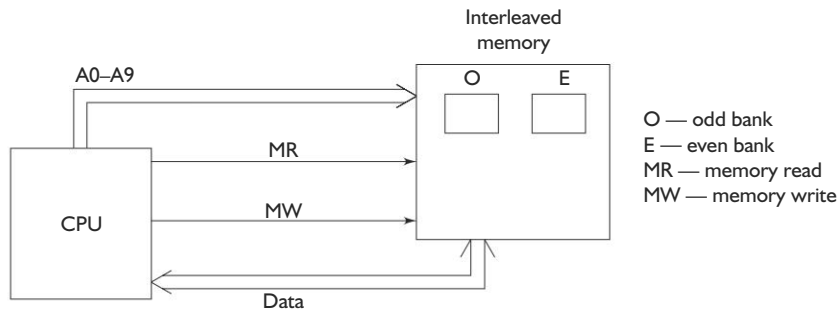


Fig. 8.2(b) CPU—Memory interface for two-way interleaving

In the case of non-interleaving, the CPU performs a memory read operation for address 1000 first and after the cycle time is over, performs read operation for address 1001 next. If the cycle time of memory is 50 ns, total access time taken for two accesses is 100 ns. Hence the bandwidth is 2 bytes/100 ns = 20 MB/sec. In a two-way interleaving, two accesses are completed in 50 ns. Hence the memory bandwidth = 2 bytes/50 ns = 40 MB/sec. Thus, a

two-way interleaving doubles the memory bandwidth. In general, the n -way memory interleaving gives a bandwidth of n times the bandwidth for non-interleaving.

Example 8.2 A CPU has 1 KB memory space. Design a four-way memory interleaving with memory ICs of 50 ns cycle time and calculate the effective bandwidth.

Address space = 1024 bytes.

Hence, no of address bits = 10.

Since four-way interleaving is followed, each module will have 256 locations. The 10 bit memory address from the CPU is used as follows:

A9–A2 : memory location within the module

A0–A1 : bank selection

Two lsbs (A0 and A1) indicate the bank. A 2-to-4 decoder is used to decode these bits and select the bank. The 8 msbs (A9–A2) are given to all the banks. Figure 8.3 shows the design at functional level.

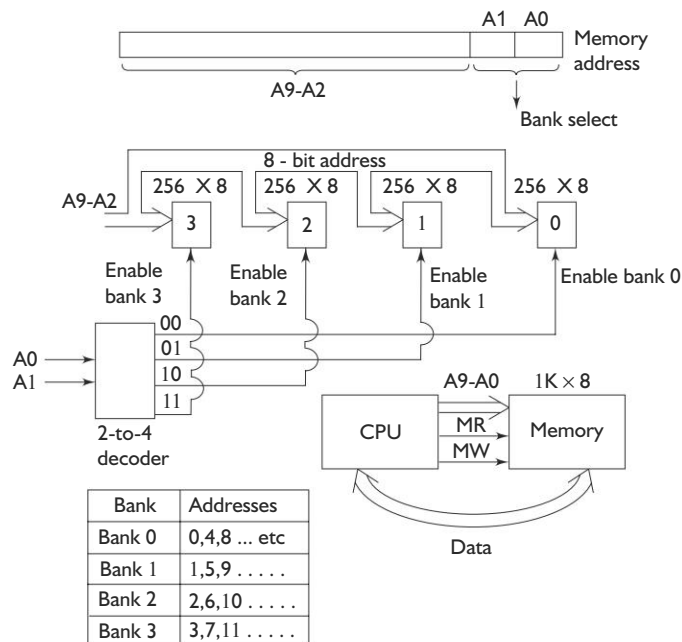


Fig. 8.3 4-way memory interleaving

8.4.1 Impact of Memory Interleaving

The n -way interleaving multiplies the bandwidth n times. It has the following drawbacks:

1. The bank decoding and enabling circuitry are needed. This increases memory cost.
2. Nominal delay is introduced by the bank decoding logic which must be taken into account while calculating the effective access time and bandwidth.
3. The CPU should have additional logic to initiate and handle a series of memory read/write cycles.

All the three drawbacks are not serious in view of performance improvement due to memory interleaving.



8.5 Write Buffer

The *write buffer* concept is functionally the opposite of the instruction prefetch. It is used to perform memory write operations on behalf of the CPU. The write buffer contains the following information:

1. Memory address where write operation has to be done.
2. Data to be written.

Usually the write buffer can store information for more than one write operation.

Figure 8.4 shows the concept of write buffer. Usually when the CPU writes a data in main memory, it could be for storing the result of an instruction. If the CPU has to be fully involved in the memory write cycle, it delays the commencement of next instruction cycle.

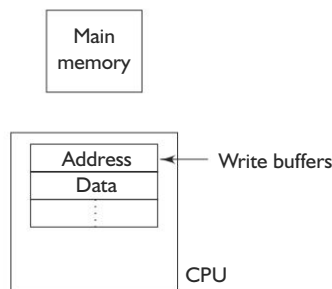


Fig. 8.4 Write buffers

If the memory is busy doing some operation initiated earlier (by DMA controller for example), it introduces waiting time after which only the write operation can be started. Hence, the precious time of CPU is wasted due to memory write operation. It is advantageous if the

CPU is freed from the memory write operation so that it commences the next instruction cycle. The write buffer is used by the CPU to dump information about the memory write cycle to be done, and the CPU proceeds further to commence the next instruction cycle. The 'write buffer control' waits for the appropriate opportunity to do write operation i.e. when memory becomes free. Then it performs the memory write cycle by taking the information from the write buffer.

What happens if the current instruction needs a memory data from a memory location (address) for which a write operation is pending in the write buffer? In such cases, allowing CPU to fetch data from main memory is meaningless. Hence, before starting any memory read operation, the write buffer is checked to see if any pending write operation for that memory address is present. If yes, the CPU does not read from main memory. Instead, it takes the data from the write buffer. This is a side benefit given by the write buffer though it may happen only occasionally.



8.6 Cache Memory

Figure 8.5 shows the usage of cache memory. The cache memory is an intermediate buffer between the CPU and the main memory. Its objective is to reduce CPU waiting time during the main memory accesses. In a system without cache memory, every main memory access results in delay in instruction processing since main memory access time is higher than processor clock period. The CPU's time is wasted during memory access (for instruction fetch, operand fetch or result storing). To minimize the waiting time of the CPU, a small but fast memory is introduced as a cache buffer between the main memory and CPU.

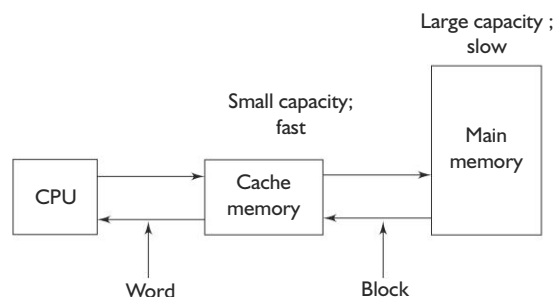


Fig. 8.5 Use of cache memory

The cache memory capacity is very small (compared to main memory) since it is a costly memory. The cache memory speed is several times better than the main memory. The

cache memory can be physically integrated with processor IC as internal cache, also known as on-chip cache.

A portion of the program and data is brought into the cache memory in advance. The cache controller keeps track of locations of the main memory which are copied in cache memory at a given time. When the CPU is in need of an instruction or operand, it receives it from the cache memory (if available) instead of accessing the main memory. Thus, the memory access is completed in a short time and the slow main memory appears as a fast memory. However, if the required item is not in cache memory, then an access to main memory is unavoidable. The item is read from main memory and put in cache memory. If the cache memory is full, some existing item is replaced. The presence of cache memory is not known to user programs and, even to the processor. The 'cache controller' hardware manages all requirements by performing necessary operations. As seen in the figure 8.5, transfer between main memory and cache memory is in blocks. Each block consists of many words. The transfer between CPU and cache memory is usually one word at a time.

8.6.1 Principle of Cache

The cache operation is based on 'locality of reference', a property inherent in programs. Most of the times, processing requirement is such that instructions or data needed are available in those main memory locations which are physically close to the current main memory location being accessed. There are two kinds of behavior pattern:

1. *Temporal locality*: The current instruction which is being fetched may be needed again soon.
2. *Spatial locality*: The adjacent instructions to current instruction may be needed soon.

In view of these two properties, while accessing the main memory for an instruction, instead of fetching just one instruction from main memory, several consecutive instructions are fetched together and stored in cache memory.

Figure 8.6(a) shows the relationship between main memory and cache memory. The main memory is conceptually divided into many blocks, each containing a fixed number of consecutive locations. While reading a location from the main memory, the content of entire block is transferred and stored in cache memory. The cache memory is organized as a number of lines (also known as blocks) and the size of each line is same as the capacity of main memory block. There are more blocks in main memory than the number of lines in cache memory. Hence, a formula (or function) is followed to systematically map any main memory block to one of the cache lines. When a main memory block is stored in cache memory, the cache line (line number) where the block has to be written is determined from the mapping function. Similarly, when the CPU reads from a main memory location, the

cache controller, using the mapping function, identifies the cache line where the main memory block is stored (if available). There are three popular methods for cache memory mapping:

1. Direct mapping
2. Associative mapping
3. Set associative mapping

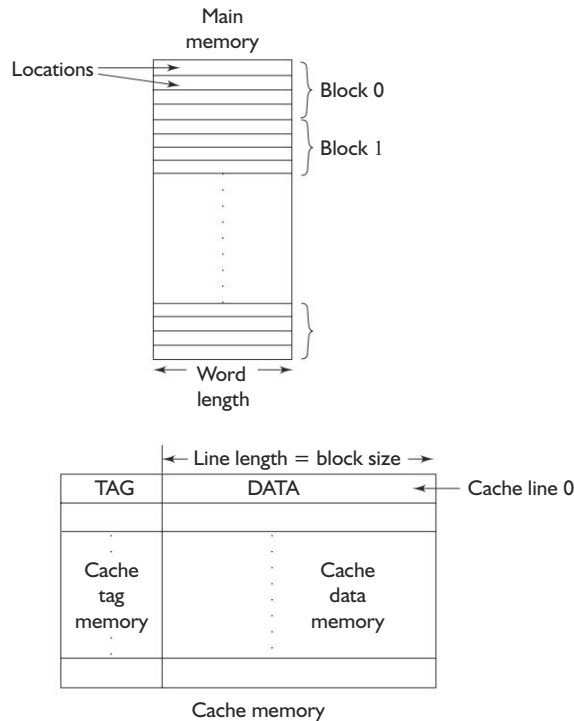


Fig. 8.6(a) Structure of cache memory

Functionally, the cache memory has two parts as shown in Figure 8.6(b):

1. Cache data memory
2. Cache tag memory

Each line of the cache data memory contains a memory block. The corresponding tag line in the cache tag memory identifies the block number of this memory block. The tag pattern is a portion of the main memory address of the block. It varies with mapping technique (direct, associative and set associative). These are discussed in forthcoming sections.

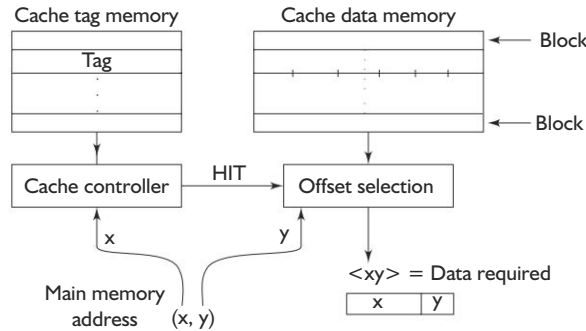


Fig. 8.6(b) Basic principle of cache operation

8.6.2 Hit and Miss

When the CPU makes a main memory access, the cache controller checks the cache memory to see whether the current memory address is already mapped onto cache. If it is mapped, the required item is available in cache memory and this situation is called 'cache hit'. The required information is read from cache memory. On the other hand, if the current main memory address is not mapped in cache memory, the required information is not available in cache memory and this situation is known as 'cache miss'. Hence, the entire block containing the main memory address is brought into cache memory. Now there are two ways of supplying the information to the processor:

1. Store the entire block in the cache memory first and then read the required item from cache memory and supply it to the processor.
2. As soon as the required item is read from the main memory, supply it to processor and then store it in cache memory.

The more frequent the cache hits are, the better it is since every 'miss' leads to accessing the main memory. The time taken to bring the required item from main memory and supply it to processor is known as 'miss penalty'. The *hit rate* (also known as hit ratio) provides the fraction of the number of accesses which faced 'cache hit' to the total number of accesses. The hit rate depends on the following factors:

1. Cache memory line size.
2. Total cache memory capacity.
3. Mapping function followed.
4. Replacement algorithm used by the operating system that decides what should be in cache memory.
5. Type of program being run.

An universal replacement algorithm which can minimize the hit rate for all situations is impossible.

8.6.3 Direct Mapping

In direct mapping, a given block of main memory is mapped to a specific line in cache memory. In other words, the cache controller has no flexibility in placing the main memory block in the cache memory lines. Figure 8.7 shows the concept of direct mapping. The main memory address is grouped into three fields: TAG, LINE, WORD. The bits in the WORD field identifies the word number within the block. The TAG and LINE fields together specify the block number. Suppose, the cache memory has c number of lines

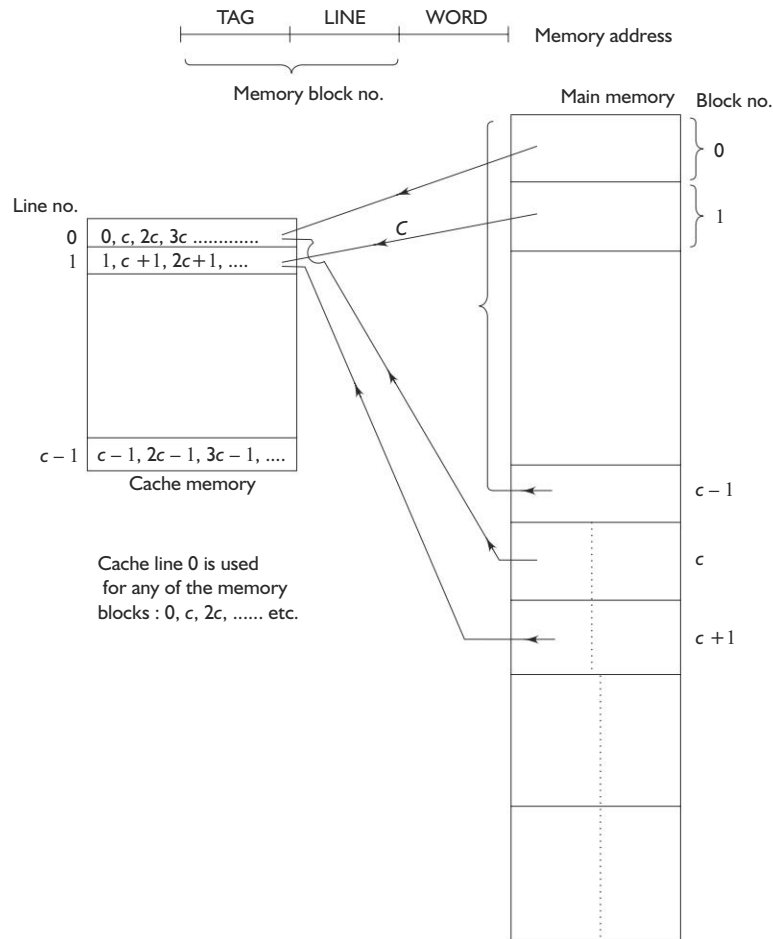


Fig. 8.7 Direct mapping

(blocks) and the main memory has m number of blocks. The direct mapping is specified as:

$$l_{cm} = b_{mm} \text{ modulo } c$$

where

$$l_{cm} = \text{cache line number,}$$

$$b_{mm} = \text{main memory block number, and}$$

$$c = \text{total number of cache lines}$$

The figure also shows the allocation of different memory blocks to different cache lines as per direct mapping. When processor performs a main memory read operation, the cache controller finds out the cache line number from the LINE field of the main memory address. The tag in that cache line is matched with the TAG field in the memory address. Now there are two possibilities:

- (A) If it matches, it is a HIT. The required data is available in the data cache in this line. Reading from that line is initiated. The required word in that block is selected using the WORD field in the memory address.
- (B) If the tag in the cache line does not match the TAG field in the memory address, it is a MISS. Hence, the cache controller initiates a main memory read operation. An entire block is transferred and stored in that cache line replacing the old block. The tag memory is also updated with the TAG field of the memory address. The required word is sent to processor.

8.6.3.1 Merits of Direct Mapping

The direct mapping is the simplest type of mapping. The hardware required is very simple since the tag of only one cache line is matched with the TAG field in the given memory address. The cache controller quickly issues hit or miss information.

8.6.3.2 Demerits of Direct Mapping

There is no flexibility in the mapping system. A given memory block is tied (mapped) to a fixed cache line. If two frequently accessed blocks happen to be mapped to same cache line (due to their memory addresses), HIT ratio is poor resulting in frequent replacement of cache lines. This slows down program execution. However, this problem is severe only for some type of programs.

Example 8.3 A 32 bit computer has 32 bit memory address. It has 8 KB of cache memory. The computer follows direct mapping. Each line size is 16 bytes. Show the memory address format and cache memory organization.

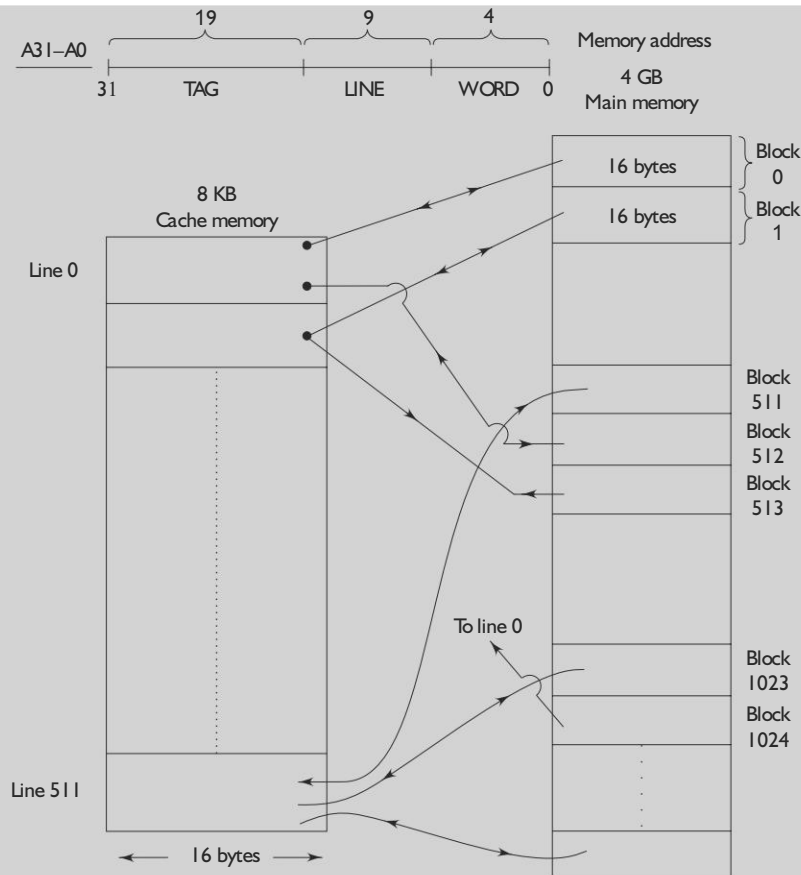


Fig. 8.8 8 KB cache and 32 bit address: Direct mapping

Figure 8.8 shows the mapping details. The memory address is 32 bits. Hence, the processor can have 4 GB of main memory. It is logically divided into 16 byte blocks. Each block has four words. Each word is 4 bytes. The four lsbs of memory address (A0–A3) give the word number within the block. 28 msbs of address (A31–A4) have to be divided into TAG field and LINE field. The LINE field length can be found first since cache memory size is given. The cache memory has 8 KB. Dividing by the line size (block size) of 16 bytes,

$$8192/16 = 512$$

Hence the number of lines in cache memory is 512. So we need 9 bits for the LINE field. The remaining 19 bits are available for TAG field.

8.6.4 Associative Mapping

The associative mapping rectifies the problem of rigidity seen in direct mapping. In associative mapping, any memory block can be mapped to any cache line. Figure 8.9 shows the principle of associative mapping. The memory address consists of two fields: TAG and WORD. The TAG field identifies the memory block number. When processor performs a memory read operation, the cache controller has to match the TAG field in the address with the TAG contents of all lines in the cache. On finding match (hit) with any line's tag, the block stored in that line is read from cache. If there is no match (miss) with all lines, the cache controller initiates memory read operation. The block read from main memory is stored in any cache line if the cache is not full. If cache is full, only then the replacement of a block in cache is required. Which block has to be replaced is decided by the replacement algorithms followed by the cache controller.

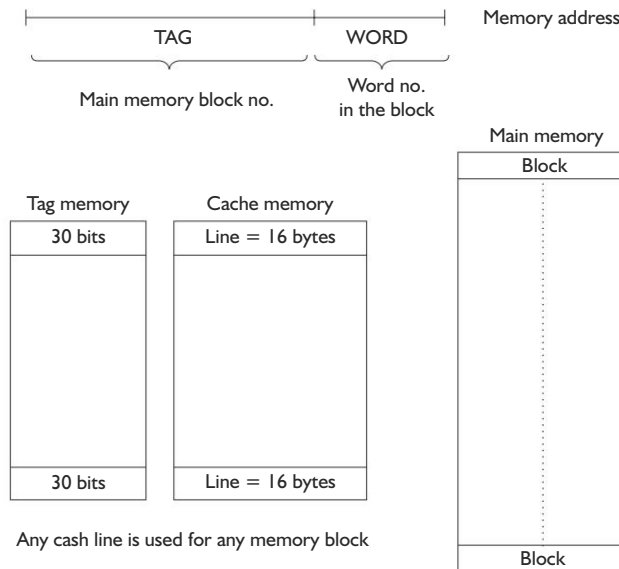


Fig. 8.9 Associate mapping

8.6.4.1 Merits of Associative Mapping

The associative mapping offers total flexibility. Any memory block can be moved to any cache line. Hence the replacement of cache block is needed only if the cache is totally full. This gives better performance since time spent on replacement is minimized. Of course, the cache size also influences the frequency of replacement. Number of lines in cache is not fixed in associative mapping.

8.6.4.2 Demerits of Associative Mapping

The associative mapping is a costly system for implementation. The cache controller becomes complex since the tags of all cache lines are matched simultaneously in order to minimize the cache controller delay. In other words, parallel searching of all cache lines is done for a specific TAG pattern. This is called an associative search. A special memory called ‘associative memory’ or Content Addressable Memory (CAM) is used for this purpose.

Example 8.4 A 32 bit computer has 32 bit memory address. The computer follows associative mapping. Each line size is 16 bytes. Show the memory address format and cache memory organization.

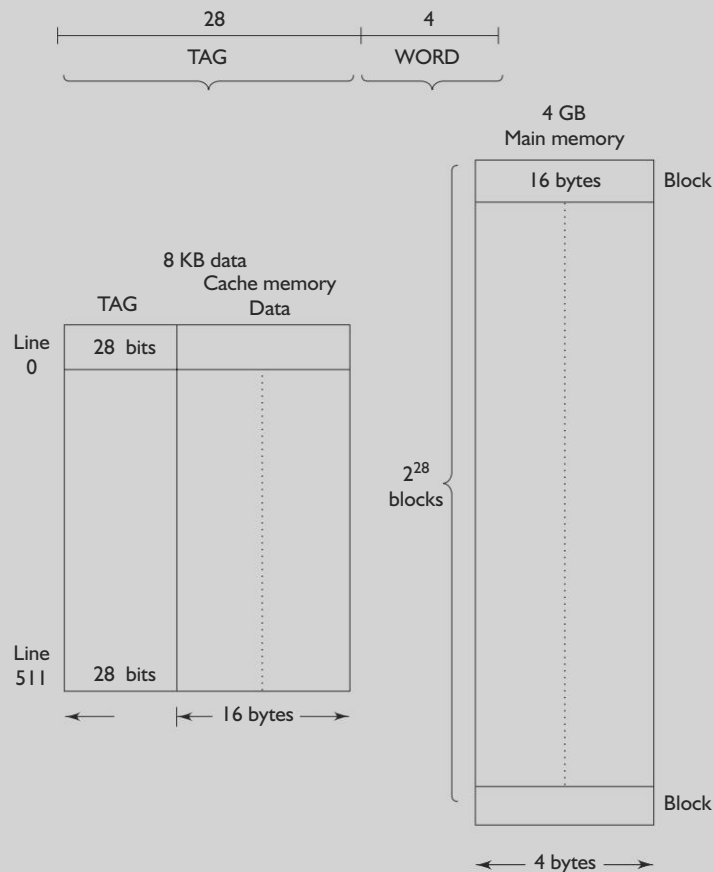


Fig. 8.10 Associative mapping — 8 KB cache and 4 GB main memory

Figure 8.10 shows the memory address format. Four lsbs (A0–A3) specify the word number within the block. The remaining 28 bits (A31–A4) give the TAG field that is nothing but the block number. There are 2^{28} blocks in memory. The number of lines in the cache memory is not dependent on any other factor. It is chosen based on performance expectation and cost. Let us assume 8 KB cache memory. Since the line size is 16 bytes, the cache memory has 512 lines.

8.6.5 Set Associative Mapping

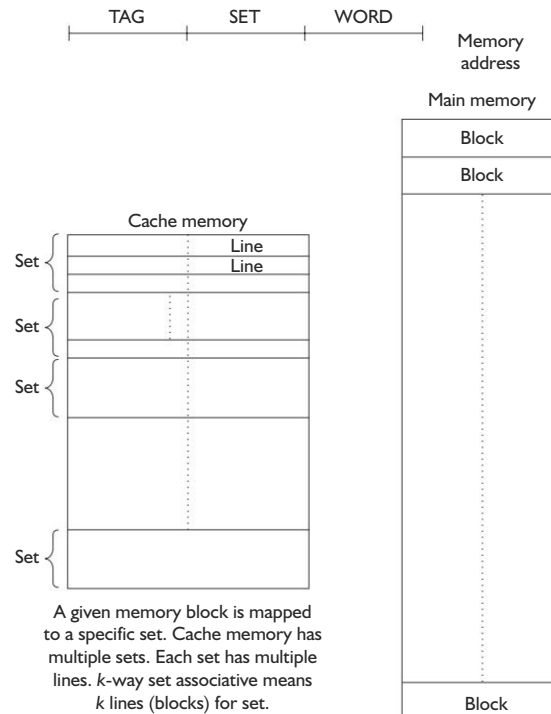
The set associative mapping combines the concept of direct mapping with associative mapping to provide a cost effective and reasonably flexible mapping scheme. The total number of cache lines are grouped into multiple sets. Each set has multiple lines of equal number. If there are k lines in each set, the mapping system is known as k -way set associative. A main memory block can be mapped into a specific set only. Within a set, the memory block can be placed in any of the k lines. The format of memory address is shown in Fig. 8.11. It has three fields: TAG, SET and WORD. The SET field provides the set number.

When the processor performs a memory read operation, the cache controller uses the SET field to access the cache. Within the set, there are k number of lines. Hence the TAG field in the address is matched with the tag contents of all k lines. On receiving a HIT, the corresponding block is read and the required word is selected. On receiving a MISS, the block is read from main memory and stored in the previously identified set. If the cache (set) is full, then replacement of one of the lines is done as per replacement algorithm.

Each set is like a small cache memory. If there is only one set, it is same as associative mapping. On the other hand, if there is only one line/set, then it is same as direct mapping.

8.6.5.1 Merits of Set Associative Mapping

1. Compared to direct mapping, multiple choices are available for mapping a memory block. The number of options depend on the size of k . Usually it is a small number: 2, 4, 8 etc. Thus, better flexibility is provided.
2. During reading, the tag matching (search) is limited to the number of lines in the set. Hence, search is only within a set unlike in associative mapping, where search is for the entire cache memory.

**Fig. 8.11** Set associative mapping**8.6.5.2 Demerits of Set Associative Mapping**

Its implementation cost is more than direct mapping but cheaper than associative mapping. However, compared to the advantages of set associative mapping, this cost increase is irrelevant.

Example 8.5 A 32 bit computer has 32 bit memory address. It has 8 KB of cache memory. The computer follows four-way set associative mapping. Each line size is 16 bytes. Show the memory address format and cache memory organization.

Four bits of address A0-A3 go for word number since block size is 16 bytes. Total number of lines in cache memory is $8 \text{ KB} / 16 = 512 = 2^9$. Since we have four-way set associative mapping, the cache is divided into 128 sets. Each set has four lines. Since there are 128 sets, 7 bits are needed for the set number. Hence SET field has 7 bits. The remaining 21 bits form the TAG field. Figure 8.12 shows the address format and mapping.

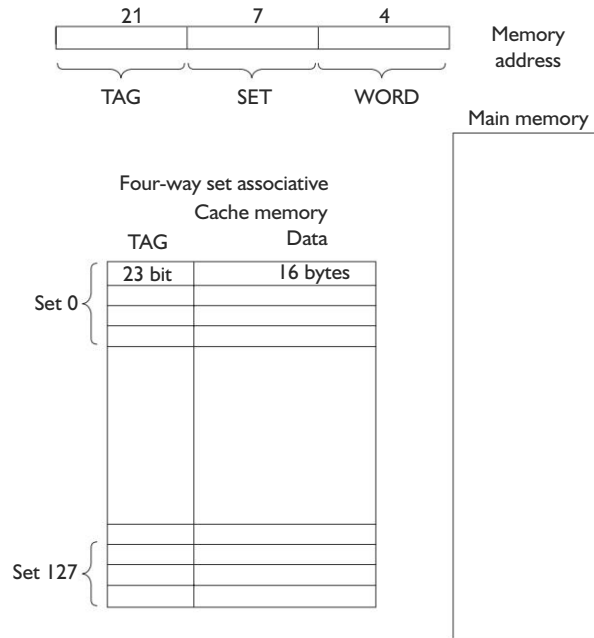


Fig. 8.12 Four-way set associative mapping

8.6.6 Cache Replacement

When an address accessed by the CPU is not mapped in cache memory, then a direct access is made to the main memory. Along with the required word, the entire block is transferred to cache memory. Suppose the cache memory is already full, then some existing contents of the cache memory are deleted to create space for the new entry. The portion of cache memory which is deleted is a matter of interest. There are special algorithms that maximize the hit rate.

8.6.6.1 Replacement Algorithms

The replacement problem is an interesting one. It is faced when the cache memory is full and a new block from main memory has to be stored in the cache memory. Obviously an existing block of cache memory has to be replaced with the new block. In the case of *direct mapping cache*, we have no choice. The new block has to be stored only in a specified cache line as per the mapping rule for the direct mapping cache. For *associative mapping* and *set associative mapping*, we need a good algorithm since there are multiple choices. If we remove a block which may have to be accessed soon by CPU, a miss penalty occurs. Hence, an

intelligent decision of selecting the cache line to be emptied is required. Certain replacement algorithms have been tried and their efficiency studied. Some of them are discussed below:

Random Choice Algorithm: This algorithm chooses the cache line randomly without any reference to its previous usage frequency. It is easy to implement with simple hardware. Though it may appear to be an ordinary algorithm, its practical efficiency is attractive.

First-In First-Out (FIFO) Algorithm: This algorithm chooses the set that has been in the cache for a long time. In other word, the block which entered the cache first, gets pushed out first. The assumption is that more recently entered items are likely to be needed more.

Least Frequently Used (LFU) algorithm: This algorithm chooses a block that has been used by the CPU minimum number of times. The assumption is that similar behaviour is likely to occur in future also.

Least Recently Used (LRU): This algorithm chooses the item that has been used by the CPU minimum number of times in the recent past. The assumption is that same behavior will continue and it may not be needed soon. Implementation of this algorithm can be done by tracking the history of usage of the items by CPU. Use of a counter for each line is a simple method of meeting the requirement.

Example 8.6 Design a circuit to implement the LRU algorithm of replacement for a four-way set associative cache memory.

Let us consider a four-way set associative cache. Figure 8.13 assumes a two bit binary counter for each block in the set. Let us call the four blocks as A, B, C and D. The counter ('age counter') for each block gives its history information. The initial values are shown in Fig. 8.13(a). The counter value indicates the extent of accesses to the block it represents. If the counter value = 00, it means the block is the most recently accessed one. If the counter value = 11, it means its block is the least recently accessed one.

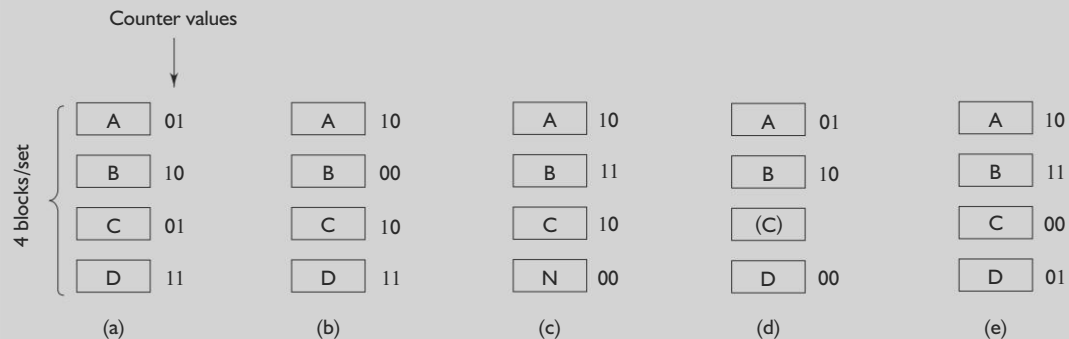


Fig. 8.13

Circuit for LRU algorithm: (a) Initial condition; cache full (b) 'HIT' for block B (c) 'MISS'; Replace D (d) Initial condition; cache not full (e) 'MISS' occupy C

Case 1: Suppose there is a cache hit for the block B. Its counter has 10 as initial value. Now its counter is reset. The contents of other counters are compared with the initial value of the block B. The counters whose values are less (than B's counter), are incremented by 1. Other counters are undisturbed. So counters of A and C are incremented whereas the counter of D is untouched. The updated status is shown in Fig. 8.13(b).

Case 2: Suppose there is cache miss when the initial status is shown in Fig. 8.13(a). Now there are two possibilities: (a) cache is full (b) cache is not full.

Let us first consider the cache full situation as in Fig. 8.13(a). On checking the values of all the counters, it is observed that D has highest value of 11. So D is the block that has not been accessed in the recent past. So D has to be replaced (moved to main memory) and the new item value enters there. Its counter is made 00. The other counters are incremented by 1. The updated values are shown in Fig. 8.13(c).

Next let us consider the case when the cache is not full as shown in Fig. 8.13(d). The new item can be stored in an empty line and its counter is made 0. The other counters are incremented by 1. Figure 8.13(e) shows the updated status.

8.6.7 Cache Write Policy

Suppose, the CPU is currently executing an ADD instruction that stores the result in the main memory location. Should the result be stored in the main memory or the cache? If the address of the result (operand) is not presently mapped in the cache, then obviously the result should be stored in the main memory. On the other hand, if the address is mapped in the cache, there are two options:

1. Write the result in both the main memory and cache memory. This policy is known as *write-through* policy.
2. Write the result in cache memory only, but mark a flag in the cache to remember that the corresponding main memory content is obsolete. In future, whenever the content in cache memory is going to be replaced, it is stored in main memory at that time. This policy is known as *write-back* policy.

8.6.7.1 Write-Through Policy

A write-through policy is simple and straightforward to implement. But it requires one main memory access every time there is a write requirement. Hence this delays program execution. Also sometimes it happens that what is stored in the main memory is only an intermediate result that is soon updated. So unnecessarily time is spent in storing that item in main

memory. However, the frequency of such wasteful writes depends on the behavior of the application program.

8.6.7.2 Write-Back Policy

The write-back policy saves time since writing in main memory is not done for every write operation time. Certain items may never be written in main memory. Only when the item has to be removed from the cache, it is written into main memory. Whether removal will happen or not, is not known in the beginning. It depends on the future instructions.

Figure 8.14(a) illustrates the sequence of operations by the cache controller for a memory read operation by the CPU. Figure 8.14(b) illustrates the sequence for a memory write operation by the CPU.

8.6.8 Snooping and Invalidation

There is a possibility of the main memory location to get updated without the knowledge of processor (and cache memory). A typical example is an input operation from hard disk when DMA controller stores the data in main memory. What happens when the DMA controller is writing in a memory location that has been mapped in cache? Assume, that the memory address 1000 contains the character M. Since address 1000 has been mapped in cache memory, this data is also available in one of the cache lines. Now suppose DMA

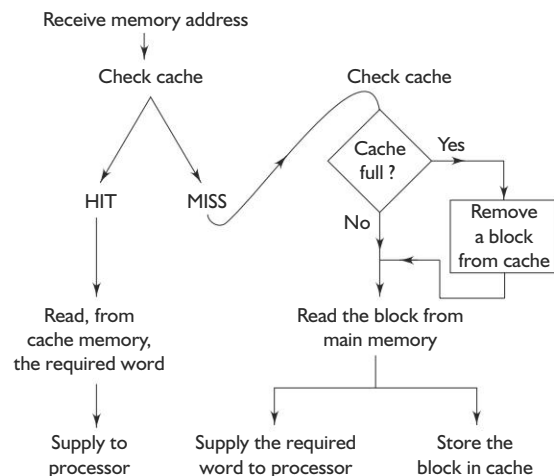
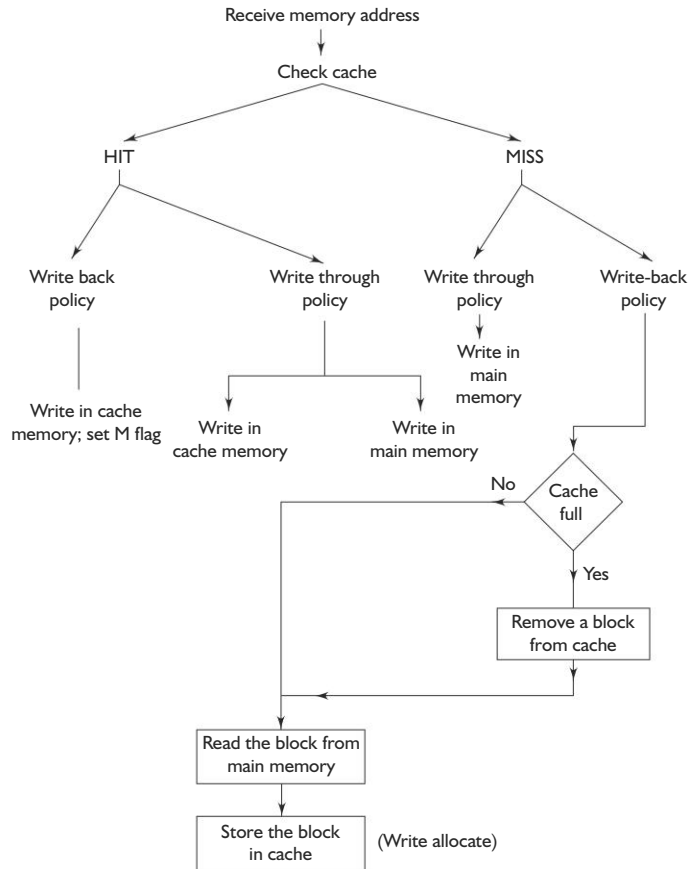


Fig. 8.14(a) Memory read sequence

**Fig. 8.14(b)** Memory write sequence

controller writes the (hard disk data) character C in main memory address 1000. If it is allowed, as it is, then the main memory location will have C whereas the corresponding cache location will have M. Since mismatch cannot be allowed between main memory and cache memory, a bus monitoring logic checks the memory (bus) activities by DMA controller (or another other processor if any). This is known as *snooping*. On detecting the commencement of a memory write sequence for an address that has been mapped in cache memory, immediately the cache controller is alerted. It is asked to invalidate the data in cache memory. For this purpose, the memory address is sent to cache controller. The cache controller sets INVALID flag for this entry. Suppose subsequently the CPU has a memory read operation for address 1000. There will be a cache hit for that address but the cache controller does not use the cache data in view of the invalid flag status. Instead, it will fetch

the new data from main memory and store it in the cache apart from supplying it to the CPU.

8.6.8.1 Cache Flush

Immediately after power-on, none of the entries in cache memory is valid since, no program has been commenced yet by the CPU. Hence for all the cache lines, invalid flag bit has to be set by the cache controller. Similarly external hardware can request the CPU to do invalidation of all cache lines. Such mass invalidation is known as *cache flushing*. Some CPUs have FLUSH instruction, while executing which cache flush operation is performed.

8.6.9 Cache Performance and Hit Rate

The design of a cache memory takes into account both cost and performance. It may appear that the bigger the cache memory, the higher is the cache performance. The major parameters of cache memory design are as follows:

1. Cache memory capacity
2. Cache line size (memory block size)
3. Set size

Cache Memory Capacity: The cost increases proportionally with capacity. Infact, there is hidden additional cost due to decoding and other control circuits. Hence, it is desired to keep the cache memory capacity as small as possible.

Cache Line Size: If the line size is large, it helps in reducing the number of main memory accesses. But a big line size reduces the number of blocks that can be accommodated in cache memory. Hence the block size should be chosen after careful study of system behavior for various block sizes.

Set Size : If there are more lines per set, hit rate is better. But too big the set size, increases cost due to increased circuitry for tag match.

8.6.9.1 Hit Rate

Ideally the hit rate should be 1 so that every main memory access by the processor is serviced by the cache memory and no main memory access is needed at all. Practically, it is impossible to achieve this and hence a hit rate close to 1 is attempted. A very poor design results in a hit rate close to 0. This means that there are too many main memory accesses. A

hit rate 0 means all main memory accesses by the processor results in MISS. Though certain design parameters help in achieving a hit rate close to 1, the behavior of application program can easily upset the hit rate.

Let us assume following parameters:

h – hit rate

$t(\text{mm})$ – main memory access time

$t(\text{cm})$ – cache memory access time

$t(\text{bm})$ – time to transfer a block from main memory to cache memory

Average system performance is given by inverse of average instruction cycle time. The instruction cycle time consists of two components:

1. Time for cache independent activities during instruction cycle.
2. Time for cache dependent accesses (main memory accesses).

The first component depends on the processor internal clock and the hardware organization. The second component depends on hit rate. Ignoring the first component for the time being, the average time spent by processor during memory accesses is given by

$$t(\text{cm}) + (1-h) \times t(\text{mb})$$

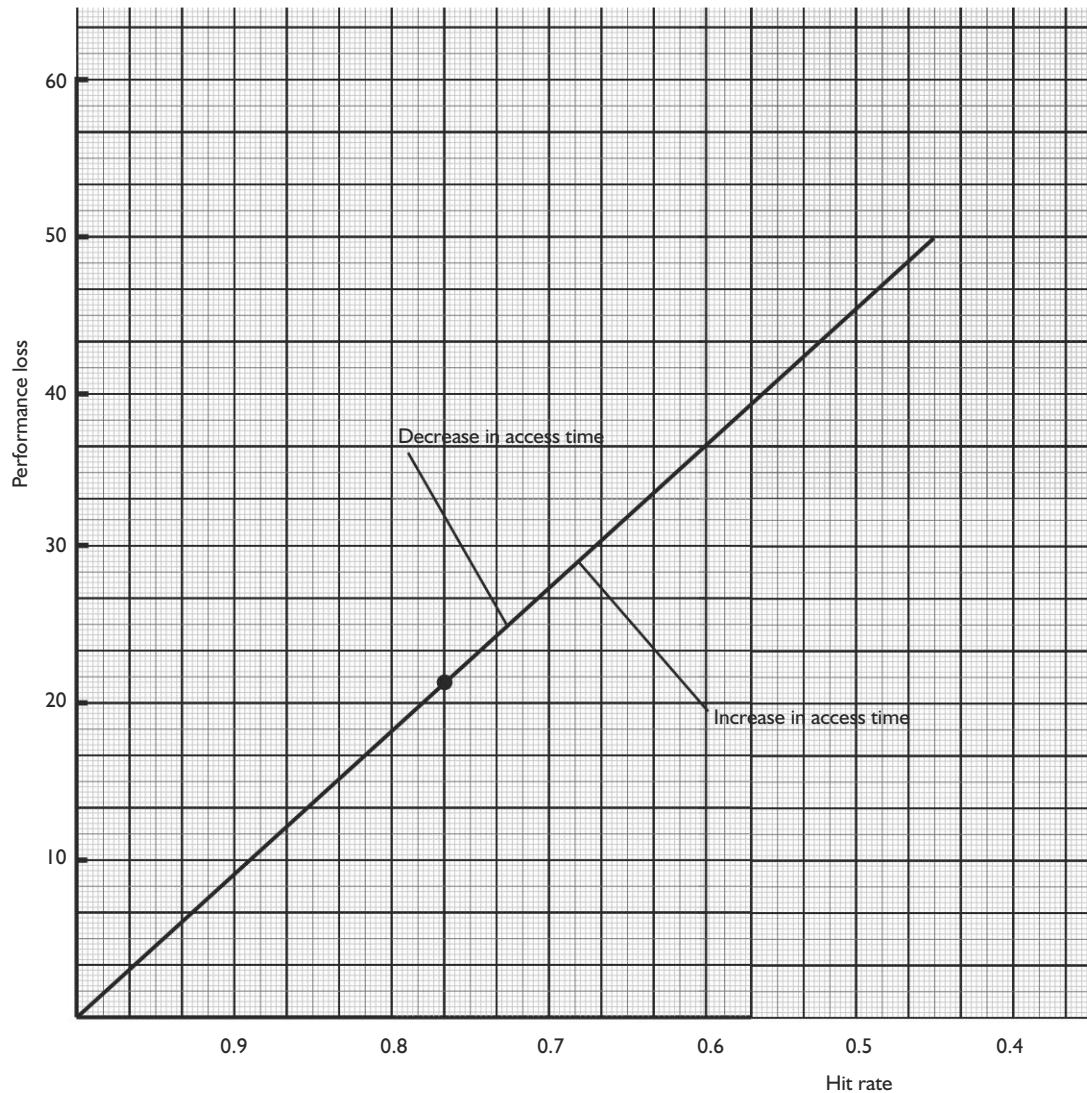
If the block size is small, $t(\text{mb})$ is small. Present day processors transfer entire block in one bus cycle. For instance, the 80486 microprocessor does burst bus cycles for this purpose. The Pentium microprocessor doubles its external datapath width for the same objective. Hence we can approximate that $t(\text{bm})$ equals to $t(\text{mm})$ and restate the average memory access time, $t(\text{ma})$ as follows:

$$t(\text{ma}) = t(\text{cm}) + (1-h) \times t(\text{mm})$$

Assuming that the cache memory is five times faster than main memory,

$$t(\text{ma}) = t(\text{cm}) + (1-h) \times 5 \times t(\text{cm})$$

Calculating for hit rates 0.98, 0.97, 0.96 we obtain the $t(\text{ma})$ values of $1.1 t(\text{cm})$, $1.15 t(\text{cm})$ and $1.2 t(\text{cm})$, respectively. This shows that a slight reduction in hit rate increases program execution time heavily. Figure 8.15 shows typical relationship between hit rate and performance. A computer design process involves a careful simulation of system behavior before finalizing cache parameters.

**Fig. 8.15** Hit rate vs performance

8.6.10 Unified and Split Cache

The cache memory is of two types:

1. *Unified (Common) Cache* memory that stores both instructions and data. The INTEL 80486 microprocessor has 8 KB unified cache as shown in Fig. 8.16.

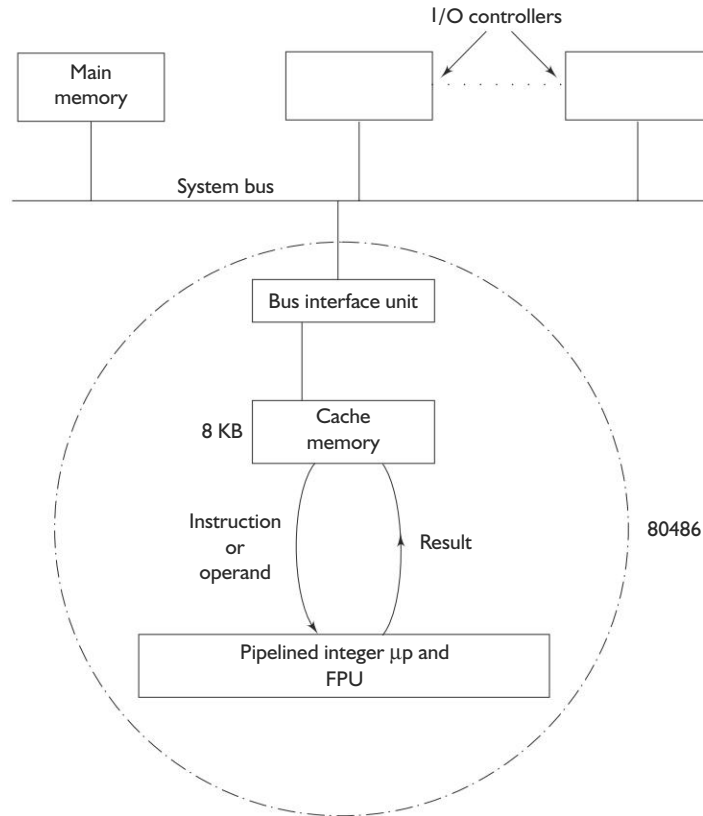


Fig. 8.16 Architecture of unified cache in 80486

2. *Split Cache* that has a separate instruction cache and data cache. INTEL Pentium microprocessor has split cache of 8 KB for instructions and 8 KB for data as shown in Fig. 8.17.

The Unified cache is easy to implement but it offers poor hit rate particularly in a pipelined CPU in which multiple memory access may be required at a time by different stages. A unified cache memory keeps some request pending, causing delay, till current access is completed. The split cache provides parallelism. Both code cache and data cache can be accessed in parallel. This reduces the wait time and improves performance.

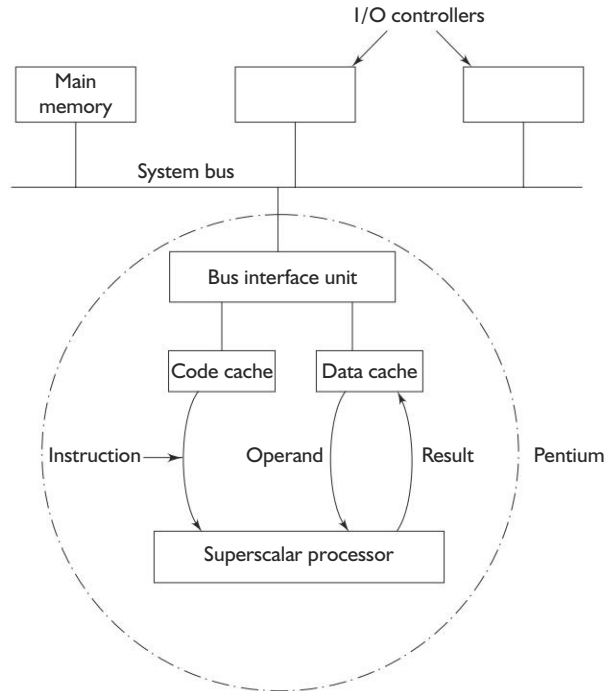


Fig. 8.17 Architecture of split cache in pentium

8.6.11 Multi-level Cache

When the main memory size is large, some computers use a two level or three level cache memory system. The cache immediately next to the processor is known as level 1 cache or primary cache. The next level cache is called a level 2 cache or secondary cache. Most microprocessors are incorporating multi-level caches on-chip. Figure 8.18(a) shows the concept of two level cache and Fig. 8.18(b) shows the architecture of two level cache in

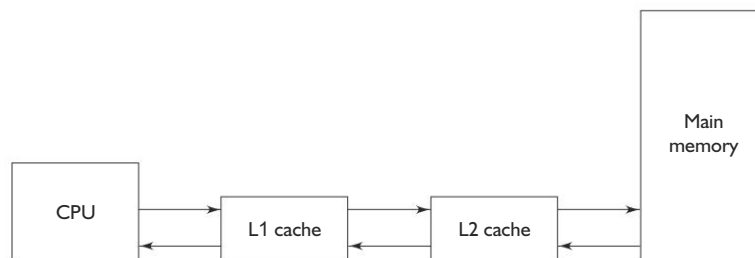


Fig. 8.18(a) Two level cache memory

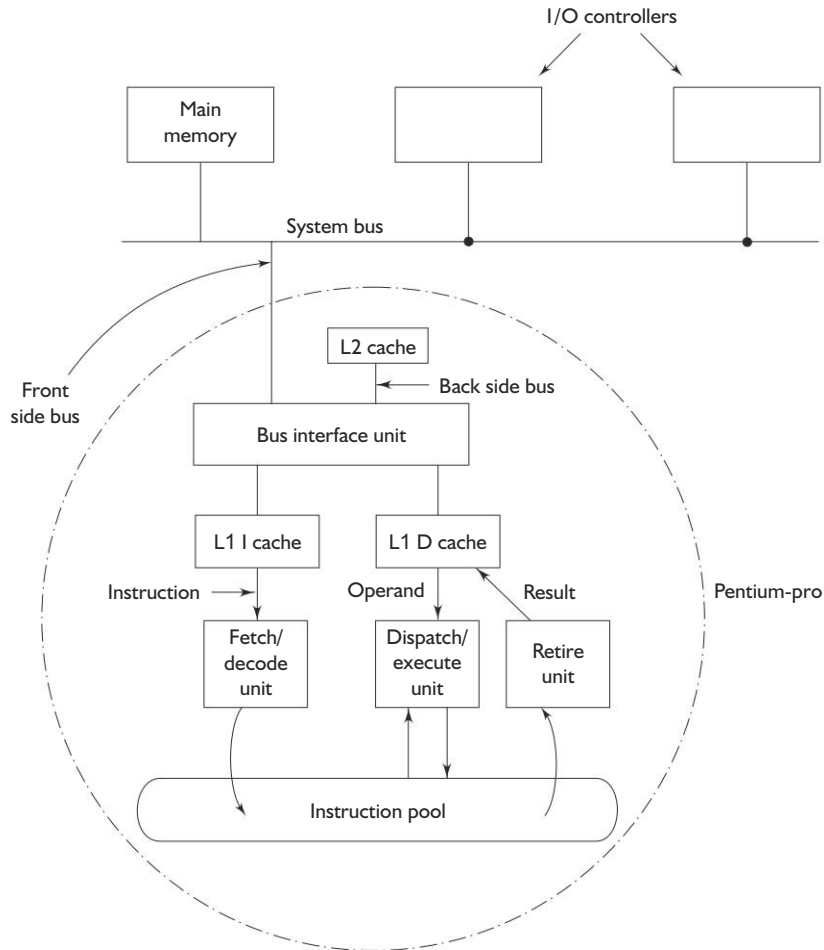
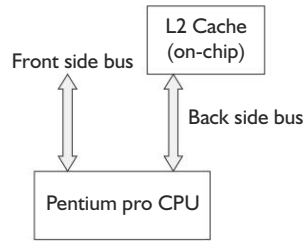


Fig. 8.18(b) Architecture of two level cache in Pentium-pro

Pentium pro microprocessor. The Pentium pro follows a dual bus architecture as shown in Fig. 8.19. There is a dedicated bus between processor and L2 cache known as backside bus. The front side bus is the system bus. The dual bus architecture provides two advantages:

1. Parallelism: simultaneous transfers on both buses.
2. High speed communication between CPU and L2 cache because of dedicated bus. This is not possible if L2 cache is also linked to system bus, because of bandwidth limitations of system bus.

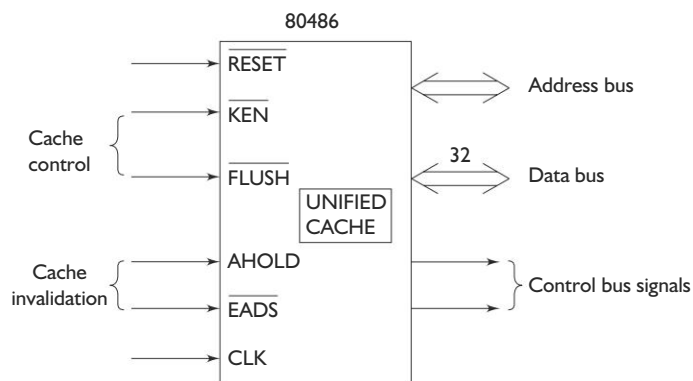
**Fig. 8.19** Dedicated cache bus

8.6.12 Cache Examples in Sample Intel Microprocessors

Intel microprocessors 80286 and 80386 can support external cache memory but do not have internal cache memory. The 80486 and subsequent microprocessors of Intel incorporate on-chip cache memory.

8.6.12.1 80486 Cache

The 80486 has a on-chip unified cache of 8 KB which is common for both code (instructions) and data. The 486 follows a four-way set associative mapping. The cache memory is organized as 128 sets and each set contains four lines. Each line is 16-bytes wide. Physically, the cache memory is split into four modules of each 2 kbytes. Each block contains 128 lines. To each block, there are 128 tags of 21 bits. The 486 cache follows ‘write-through’ strategy. Cache allocations are not made on write misses. Figure 8.20 shows a partial pinout of 80486.

**Fig. 8.20** Partial pinout of 80486

Cache Controls

The CD and NW bits in the control register CR0 provide the cache control. The CD bit enables or disables the cache. The NW bit controls memory write-through and invalidates. There are four operating modes for the cache as defined by the combinations of the CD and NW bits as shown in Table 8.1.

TABLE 8.1 Combinations of CD and NW bits

CD	NW	Operating Mode
0	0	Cache fills enabled; write-through and invalidation enabled; this is the normal operating mode
0	1	This combination is invalid since it implies a non-transparent write-back cache whereas the 486 cache is a write-through cache; this pattern if loaded will result in generation of a General Protection Fault with error code of 0
1	0	Cache fills are disabled but write-through and invalidation are enabled; this mode is used by the software to disable the cache for a short interval and then enable it without flushing the original contents
1	1	In this mode, cache fills are disabled; also write-through and invalidation are disabled the cache is completely disabled by setting both the CD and NW bits as 1's and then flushing the cache; (Diagnostic programs use this mode to monitor all memory cycles externally). If the cache is not flushed during memory read operations, cache hits may occur and data (wrong data) will be read from the cache

Cache Line Fills

Any main memory area can be cached inside the 486. However, either software or external hardware can define some portions of memory as non-cacheable. Software prevents caching pages by setting the PCD bit in the page table entry. External hardware makes the $\overline{\text{KEN}}$ (cache enable) pin inactive, during a memory access to inform the 486 that the memory address is non-cacheable.

When the microprocessor performs a read operation, if the address is (mapped) in the cache, a cache hit occurs and the data will be supplied from the on-chip cache. If the address is not (mapped) in the cache, the microprocessor has to perform an external memory read operation. If the address belongs to cacheable portion of memory the 486 performs a cache line fill. During a line fill, 16-bytes are read into the 486 and a line fill occurs.

Cache fills occur only for cache-miss during read operations. If cache-miss occurs during a write operation, no change occurs as far as the cache is concerned. If a cache hit occurs during a memory write operation, the line is updated.

Cache Line Invalidations

Cache line invalidations are needed in order to indicate that certain internal cache contents are different from the external memory contents. The hardware has a mechanism to check the write operation for the external memory by other units. When another processor or DMA controller is writing to a section of external memory, the 486 invalidates the corresponding internal contents.

There are two steps during invalidation cycle. First, the external hardware activates AHOLD (Address hold) signal. In response to this, the 486 immediately releases the address bus. Next, the external hardware activates $\overline{\text{EADS}}$ (Valid external address) signal indicating that a valid address is present on the 486's address bus. The 486 reads the address from the address pins. It then checks whether this address is mapped in the internal cache. If the address is mapped in the internal cache, then the cache entry is invalidated.

The external hardware normally activates $\overline{\text{EADS}}$ signal, when an external 'master' issues an address on the address bus. If $\overline{\text{EADS}}$ alone is issued without AHOLD signal, the 486 itself releases the invalidation address.

The valid bit for each line indicates whether a line is valid or non-valid. All valid bits are cleared when the 486 is reset or when the cache is flushed.

Cache Replacement

When the 486 adds a line in the internal cache, it first checks whether an invalid line exists in the set. If an invalid line exists in the set (out of four lines), the invalid line is replaced with the new line. If all the four lines in the set are valid, a pseudo least recently-used mechanism is used to find which line should be replaced.

Cache Flushing

The on-chip cache can be flushed by external hardware or software. Flushing the cache results in clearing all valid bits for all lines in the cache.

The external hardware activates the $\overline{\text{FLUSH}}$ pin for flushing the cache. The software uses INVD or WBINVD instructions to flush the cache. When these instructions are executed, external caches connected to the 486 are also signaled to flush their contents. For a

WBINVD instruction, the external write-back cache should write back 'dirty' lines with modified flags, (in main memory) before flushing its contents. The external cache is signaled using the bus cycle definition pins and byte enables. As far as the internal cache is concerned, the effect of the INVD and WBINVD instructions are same.

8.6.12.2 Cache Invalidation in Pentium

Figure 8.21 shows a partial pinout of Pentium processor. The Pentium cache invalidation sequence is more involved than that of 486; the Pentium microprocessor follows write-back policy. Cache invalidation is performed by the following two steps:

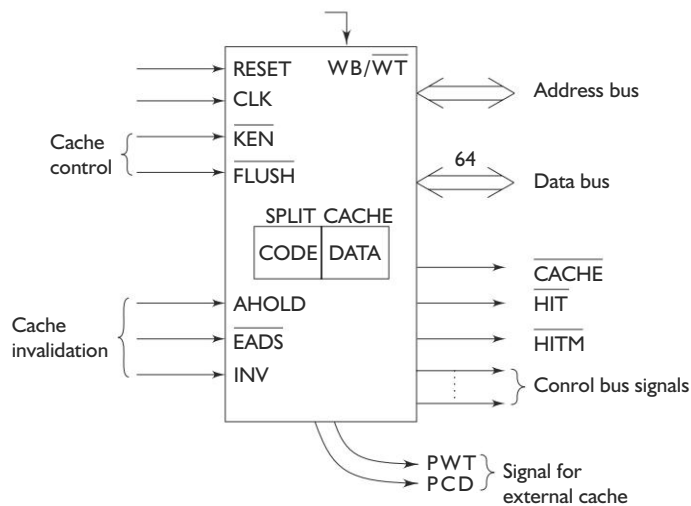


Fig. 8.21 Partial pinout of Pentium

1. The external hardware sends AHOLD signal to Pentium processor and in response, the processor releases the address bus by tristating its address pins. Then the external hardware puts the memory address on these pins and informs this to Pentium by means of the EADS signal.
2. Thereafter, the processor performs 'inquire' cycle and indicates the results by means of $\overline{\text{HIT}}$ and $\overline{\text{HITM}}$ signals. The $\overline{\text{HIT}}$ indicates the 'hit' or 'miss' status. The $\overline{\text{HITM}}$ reports that the hit is for a modified line in the data cache (which implies that the content of a cache entry is not yet stored in the corresponding location of main memory). Now the external hardware should take appropriate action. Issuing the signal INV invalidates the cache line. In case of 486, there is no inquire cycle and once the $\overline{\text{EADS}}$ signal is received, the 486 starts invalidation.



8.7 Virtual Memory

Virtual memory is desirable in the following two cases, to execute large programs:

1. The logical memory space of the processor is not sufficient.
2. The physical main memory size is kept small to reduce the cost though the processor has large logical memory space.

In olden days, the first case was responsible for having virtual memory since the computers in those times offered limited main memory space. Nowadays a large memory space is provided by the processors and hence the second case is the reason for having virtual memory.

In a virtual memory system, the operating system automatically manages the long programs without manual intervention. The user program can be larger than the physical main memory capacity. The operating system stores the entire program on a hard disk whose capacity is much larger than the physical memory size. At a given time, only some portions of the program are brought into the main memory from the hard disk. As and when needed, a portion which is presently not in main memory is brought from the hard disk, and at the same time, a portion of the program which is presently in the main memory is taken out of the main memory and stored on the hard disk. This process is known as *swapping*. In a virtual memory system, when a program is executed, swapping is performed several times.

In order to support virtual memory, the program can not address physical memory directly. When referring to an operand (data) or instruction, it provides the logical address and the virtual memory hardware translates it into the equivalent physical memory address during execution. The CPU fetches instruction or data from the main memory. Whenever the required instruction or data is not available in the main memory, the hardware raises a special interrupt known as *virtual memory interrupt* or *page fault*. The page fault is similar to the interrupt but it can occur within an instruction cycle. In response, the operating system loads a section of the program (containing the required instruction or data) from the hard disk drive to the main memory. After the page fault interrupt is serviced, the CPU continues processing the partially executed instruction. Figure 8.22 illustrates the virtual memory concept. The *address translation* is done by the Memory Management Unit (MMU).

Several methods are used for address translation which is also known as mapping. The *Associative Mapping Scheme*, *Mapping by Address Scheme* and *Segment Map Table Scheme* are three popular methods. In the *Associative Mapping Scheme*, a special memory known as ‘*associative*

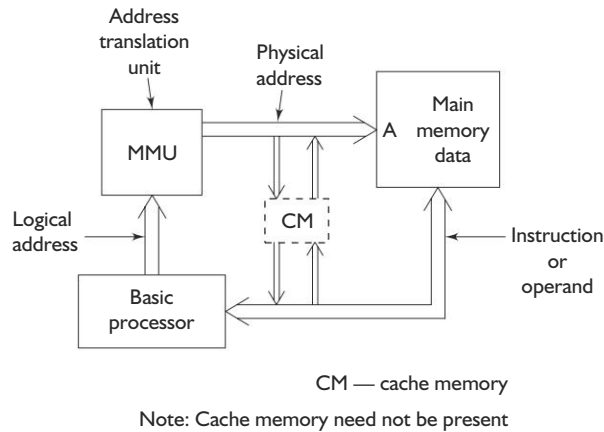


Fig. 8.22 Virtual memory concept

memory or *content address memory* is used. Each entry in this memory contains a logical address and an equivalent physical address. The addressing of this memory is done by using the logical address.

8.7.1 Advantage of Virtual Memory

1. Program size is not limited by physical memory size.
2. The user need not estimate memory allocation. Main memory allocation is done automatically according to the demands of the program.
3. Manual folding (over lay) is eliminated to run large programs.
4. The program can be loaded in any area of physical memory since the program does not use physical address

8.7.2 Virtual Memory Mechanism

A CPU's address space is the valid range of addresses that can be generated by its instruction set. The number of bits in the memory address register determines the size of the address space. Physical addresses denote actual locations in the main memory. A logical address is a memory specification used by the CPU. In many computers, logical and physical address are the same. In the virtual memory system, the logical addresses are different from the physical addresses.

The logical address has two parts. The first part identifies the module number (segment number or page number) of the program. The second part provides the offset or word

number, in the program module from the beginning of the module. A table is maintained in the hardware to indicate the portions (parts or modules) of the program which are presently available in the main memory. It contains two basic information:

1. The portions (parts) of the program that are currently in main memory.
2. The main memory area where the currently available portions (parts) are stored.

Since the program addresses the operand or instruction using a logical address, the virtual memory hardware translates the logical address into physical address before execution. Whenever an instruction or operand has to be accessed, the CPU searches the table to find whether it is available in the main memory or not. If it is available, the MMU converts the logical address into the equivalent physical address and then the main memory access is made. On the other hand, if the instruction or operand is not available in main memory, an interrupt is raised. This invokes the operating system which performs swapping. When swapping takes place, the operating system updates the table. Now the address translation is carried out and the main memory is accessed. Figure 8.23 shows the principle of address translation.

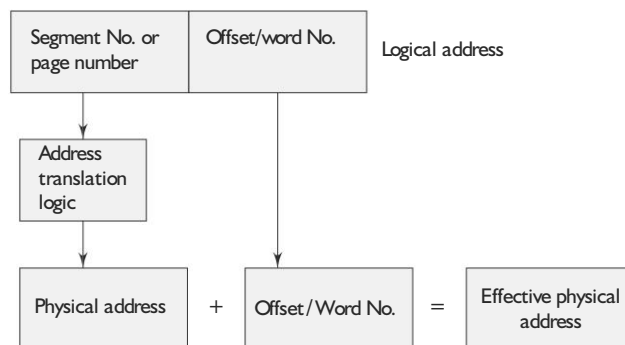


Fig. 8.23 Virtual memory mechanism

There are two popular methods in virtual memory implementation:

1. Paging
2. Segmentation

In *Paging*, the system software divides the program into *pages*. This is not known to the programmer. All the pages in a program are of same size. As a common practice, the user program is split into multiple pages of equal size. At a given time, only some pages are in the main memory. In a *multiprogramming system*, at a given time, few pages of different programs are available in the main memory. The remaining pages of different programs are stored on a hard disk which serves as the virtual memory.

In *Segmentation*, the (machine language) programmer organizes the program into different *segments*. It is a logical division of the program into meaningful modules. Different segments are not of same size.

IBM 360/67 and Control Data Cyber 180 are some of the early computer systems which offer virtual memory. Intel microprocessors 80286 onwards support virtual memory. The 80286 follows the segmentation scheme for virtual memory. The 80386 adopts both segmentation and paging schemes for virtual memory.

8.7.3 Segmentation Scheme in Intel 80286

The 80286 supports easy implementation of virtual memory by providing two hardware features:

1. Segment not present exception (virtual memory interrupt)
2. Restartable instructions

The 80286 has two operating modes:

1. 8086 real address mode (real mode)
2. Protected virtual address mode (protected mode)

The 80286 provides the real mode in order to give a downward compatibility with the 8086/8088. When programs developed for the 8086/8088 are executed on a 80286 based system, the 80286 is operated in the real mode. Multiprogramming, memory protection and use of virtual memory are not possible in real mode. In the protected address mode, the 80286 offers multiprogramming, virtual memory and memory protection. When the 80286 is reset, it enters the real mode. It begins instruction execution from the address¹ FFFFF0. To shift the processor to the protected mode, the operating system uses a special instruction—Load Machine Status Word (LMSW) with Protection Enable (PE) bit, in the machine status word, as ‘1’.

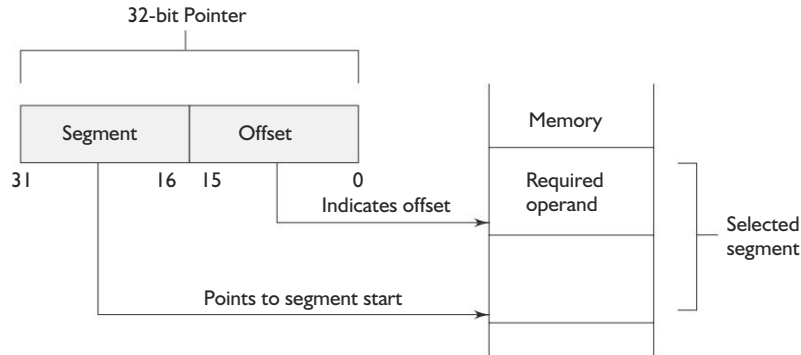
8.7.3.1 Real Mode Memory Addressing in 80186

In the real mode, the 80286 behaves similar to an 8088 (or 8086). The programs use real addresses. When a program (an instruction) addresses memory, there are two components:

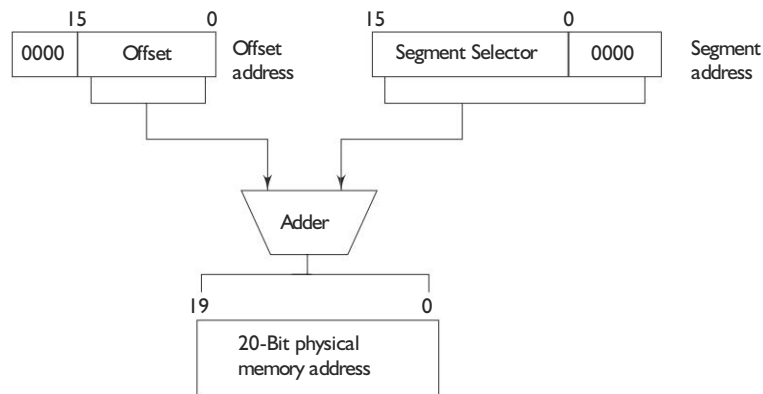
1. Segment selector
2. Offset

The *Segment selector* part indicates the desired segment in memory. It provides the segment start address. The *Offset* indicates the desired byte address within the segment. Figure 8.24 shows the principle of memory addressing either for memory read or memory

¹ All addresses are in hexa decimal unless mentioned otherwise.

**Fig. 8.24** Principle of memory addressing by 80286

write and Fig. 8.25 shows the mechanism of real mode addressing. During a memory access, the 80286 converts the 16 bit segment selector (i.e., content of segment register) into 20 bit address by shifting left by four bits. Then it adds the offset to this. Thus, the 80286 generates 20-bit physical addresses and the maximum memory range is 1 MB.

**Fig. 8.25** Memory addressing in real mode

8.7.3.2 Protected Mode Memory Addressing (Virtual Memory) in 80286

In protected mode, the memory addressing is up to 16 MB since the 80286 generates 24-bit addresses. In addition, it offers a 1 GB virtual address space, for each task, which is mapped by the 80286, into the 16 MB real address space. Programs written for the protected mode use virtual addresses. The protection mode isolates the different user programs by providing memory protection which also ensures privacy of each task's program and data. Figure 8.26 shows the mechanism of memory addressing in protected mode. In protected mode,

the selector does not directly provide the segment start address. Instead it points to a memory location where the segment start address is stored. This location is a part of segment descriptor for the segment.

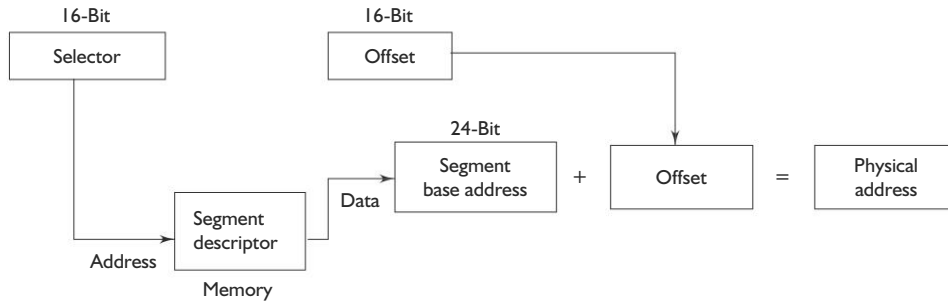


Fig. 8.26 Memory addressing in protected mode in 80286

8.7.3.3 Overview of Segmentation and Virtual Memory Mechanism in 80286

Figure 8.27 illustrates the address translation mechanism of 80286. The program gives the logical address as a 32-bit pointer. There are two parts to the pointer:

1. A 16 bit selector
2. A 16 bit offset

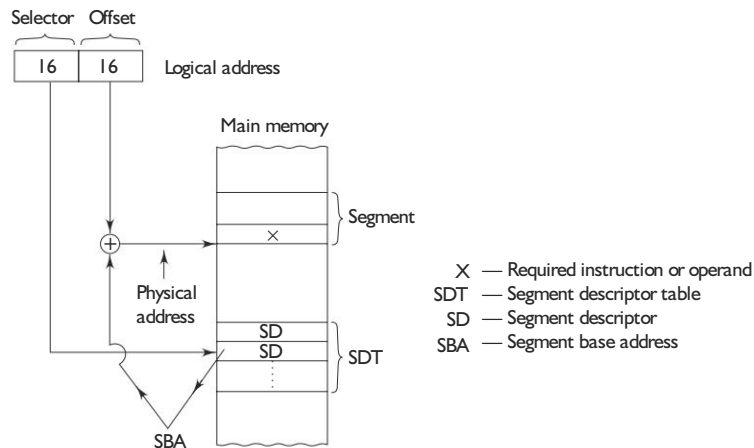


Fig. 8.27 Segmentation method of address translation

The *Selector* (in the segment register) provides an index to a memory resident table known as *Segment Descriptor Table* (SDT) that contains segment descriptors for all the segments. The

segment descriptor maintains the segment base (start) address. It also has other attributes for the segment such as segment size, access rights (protection types) and status of presence or absence in main memory. The selector is used to access the segment descriptor (in the SDT) of the currently required segment and acquire the segment base address if the required segment is in the main memory currently. To the segment base address, the 16 bit offset in the logical address is added to get the physical address. If the required segment is not available in main memory, it is known from the segment descriptor. Hence when the SDT is accessed, an interrupt known as '*segment not present*' exception is generated by the MMU. The processor switches to the operating system that loads the required segment, from secondary memory to main memory, and updates the segment descriptor in the SDT.

The readers would be shocked to notice that every main memory access needs segment descriptor which itself is in main memory. This additional main memory access (read) is for getting the segment base address. This causes heavy penalty to the instruction execution time. However, a technique is used to eliminate this penalty in most of the accesses by incorporating a 'cache' inside the processor. The 80286 has a 'segment descriptor cache' register for each of the four types of segments (code, data, stack and extra). As soon a selector value is loaded into the segment register, the 80286 automatically reads the segment descriptor from SDT in main memory and loads into the corresponding 'segment descriptor cache' register as an advance preparation. For all subsequent memory accesses within the segment, the processor reads from the internal 'segment descriptor cache' register and not from the SDT in main memory. Thus, a small cache inside the MMU accelerates the address translation process. Figure 8.28 shows the four segment descriptor cache registers corresponding to four segments. It is obvious that these are purely internal registers that are not accessible by programs.

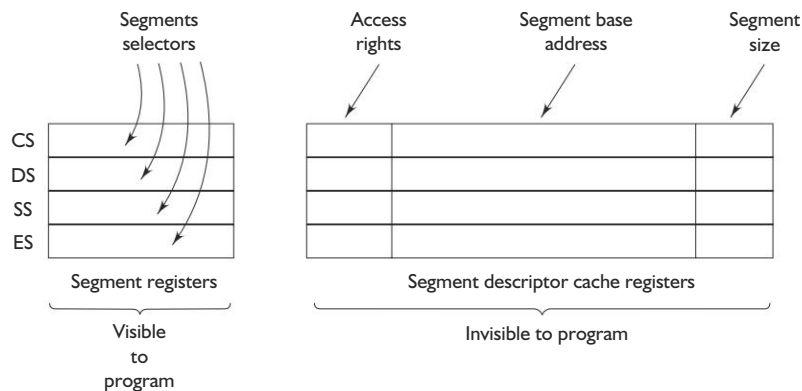


Fig .8.28 Use of 'Descriptor cache'

8.7.4 80386 Virtual Memory Mechanism

The 80386 has three operating modes:

1. Real address mode (real mode)
2. Protected mode
3. Virtual 8086 mode (V86 mode)

Real Address Mode: This mode emulates the 8086 processor, with a few additional features (such as the ability to come out of this mode). It is similar to the 286 real mode.

Protected Mode: In this mode, the entire instruction set and features of the 386 are available. The major features are multitasking, memory protection, paging and privilege operations.

Virtual 8086 Mode: This is a special mode within the protected mode. This mode offers protection and memory management (unlike the real address mode) for programs developed for the 8088. The processor can enter the virtual 8086 mode from the protected mode to run a program written for the 8086 processor, then leave the virtual 8086 mode and re-enter the protected mode to continue a program which uses the 32-bit instruction set.

8.7.4.1 Address Spaces for 386

There are three address spaces for 386:

1. Logical address space
2. Linear address space
3. Physical address space

Machine language programs (object code) use logical address for both instruction and operand address. Segmentation unit translates logical address into 32-bit linear address space. When paging unit is enabled, paging unit translates the linear address space into physical address space. If the paging unit is not enabled, the linear address corresponds to the physical address. Figure 8.29 illustrates the address translation used by 386.

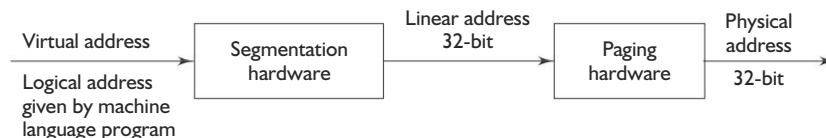


Fig. 8.29 Segmentation and Paging-80386

Figure 8.30 shows the address formats. The logical address has two parts: (a) selector (b) offset. The selector is available in the segment register. The offset is calculated by the processor by adding base, index and displacement fields as per the addressing mode. The

segmentation unit translates the logical address into linear address. If paging is enabled, the paging unit translates the linear address into physical address. If paging is disabled, the logical address is same as physical address. Figure 8.31 shows the two level address translation. The first level translation is similar to segmentation mechanism in 80286.

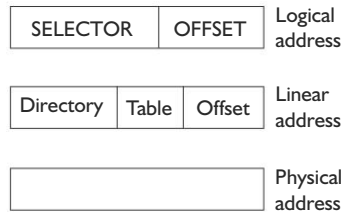


Fig. 8.30 Address formats for 386

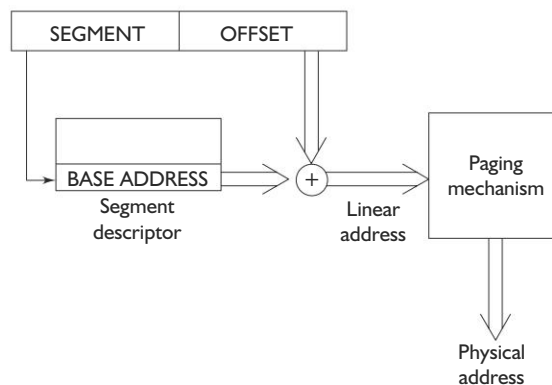


Fig. 8.31 Two level address translation in 80386/80486

8.7.4.2 Paging Mechanism

The input to paging mechanism is the linear address. All the three fields in the linear address are used as offsets by the paging mechanism. The paging mechanism uses two levels of tables to translate the linear address into physical address. These table are:

1. *Page directory* that contains start addresses of page tables and control information for them.
2. *Page table* that contains start addresses of page frames and control information for them.

The page size is 4 KB and all the tables are also of 4 KB. Figure 8.32 shows the paging mechanism. The CR3 register contains physical start address of page directory. To this address, the 'directory' field in the linear address is added. The result points to an entry in the page directory. Each entry in the page directory contains the physical start address of page table. Hence by accessing page directory, start address of currently required page table is obtained. To this address, the 'table' field in the linear address is added. The result points to an entry in the page table. This entry provides the physical start address of the page frame. To this address, the 'offset' in the linear address is added. The result is the 32 bit physical address.

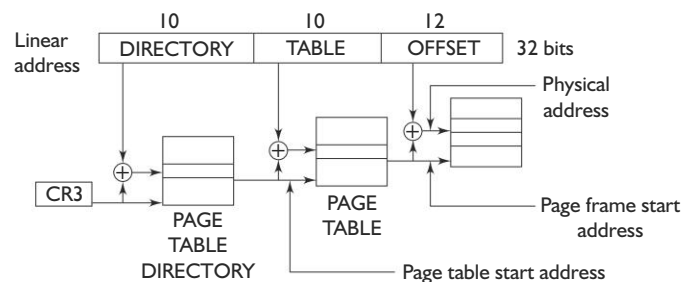
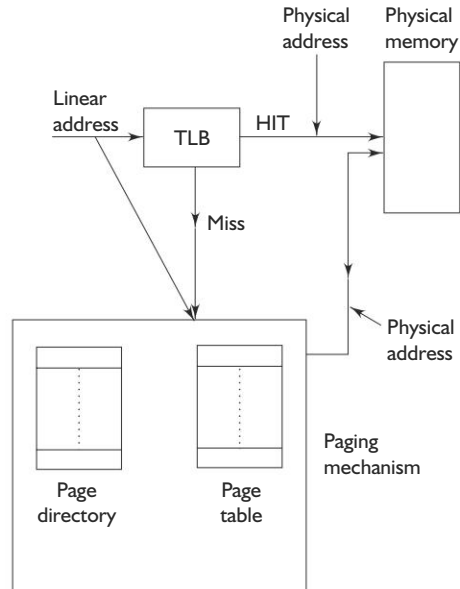


Fig. 8.32 Paging mechanism in 80386/80486

8.7.4.3 Translation Lookaside Buffer

As there are two levels of table reference needed to determine the physical address, two memory accesses are needed since all the tables reside in main memory. This will degrade the performance severely if implemented as it is. To resolve this issue, the 80386 microprocessor has a dedicated cache known as Translation Lookaside Buffer (TLB). This is a 32 entry cache memory. Each entry stores a page table entry. The TLB mechanism automatically retains the most commonly used 32 page table entries. Since each page is of 4 KB size, 128 KB of physical addresses are covered by the 32 entry TLB. The TLB in 80386 gives a hit rate of 95%, whereas the TLB of 80486 gives a hit rate of 98%. The TLB is a four-way set associative cache memory. Figure 8.33 shows the use of TLB. From the linear address, the TLB gives a hit or miss. In the case of hit, the TLB supplies page frame address. In the case of miss, the paging mechanism is activated. In case the required page is not in main memory, a 'page fault' occurs as an interrupt. The page fault can also occur for the page directory or page table. The operating system services 'page fault' by bringing required entries from secondary memory to main memory; it also updates the tables, accordingly.

**Fig. 8.33** Use of TLB

8.7.4.4 Role of Operating System

The paging system in 80386 is known as *demand paging* since a page is supplied to the main memory when required. The operating system performs the following tasks:

1. Setting up the page tables
2. Setting up page directory
3. Loading the register CR3
4. Flushing the TLB whenever a change is made to any of the page table entries
5. Implement a swapping policy (replacement algorithm)
6. Handling page faults

8.8 Associative Memory

The associative memory is also known as Content Addressable Memory (CAM). It differs from other memories such as RAM, ROM, disk etc. in the nature of accessing the memory and reading the content. To any other memory, the address is given as an input for reading the content of a location. In the case of associative memory, usually a part of the content or

the entire content is used for accessing the content. Given an argument as input, the cache memory can search and report whether it has this argument in any location. The associative memory searches all the locations simultaneously in parallel. It is useful as a cache memory wherein a search of the locations is needed to match with a given TAG.

Figure 8.34 illustrates a block diagram of an associative memory of m locations of n bits each. While writing into the associative memory, no address is used. The associative memory finds an empty location for storing the input data. To perform a search operation, the argument (e.g. TAG) is given to the argument register which has n bits. Two types of searches are possible:

1. Searching on the entire argument
2. Searching on a part (field) within the argument

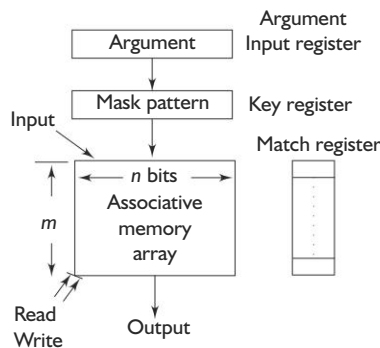


Fig. 8.34 Block diagram of an associative memory

The key register is supplied a mask pattern that indicates the bits of argument to be included for search pattern. If any bit is 0 in the mask pattern, then the corresponding bits of the argument register should not be included in the argument search. For an entire argument search, the key register is supplied all 1's. While reading (searching), the associative memory locates all the words which match the given argument and marks them by setting the corresponding bits in the match register. Subsequently, reading from all these locations is performed by sequential access to the matched words.

The associative memory is costly since it requires matching logic with each cell. In view of its high speed search capability, it is the best choice for cache memory and TLB in address translation unit.

SUMMARY

Two parameters of the main memory are of utmost importance to a computer: speed and capacity. The speed of the main memory is slower than that of the processor. The physical capacity of the main memory is limited due to the cost factor. The virtual memory technique helps to run lengthy programs with limited main memory space whereas the speed mismatch is resolved by following techniques: instruction prefetch, memory interleaving, write buffer and cache memory.

The objective of the instruction prefetch is to receive one or more instructions from the main memory, in advance, while the current instruction is being executed. A set of 'prefetch buffers' store the instructions that are fetched in advance. The instruction prefetch is a hardware feature and the program is not aware of it.

Memory interleaving is a technique of reorganizing the main memory into independent modules so that the overall bandwidth is increased many folds. The n-way memory interleaving provides a bandwidth of n times the bandwidth of non-interleaving case.

Functionally, the write buffer is the opposite of the instruction prefetch. It relieves the CPU from performing memory write. The CPU can store information about the memory write cycle in the write buffer and proceeds with the next instruction cycle. The write buffer waits till the memory is free and performs memory write cycle by taking the information from the write buffer.

The cache memory acts as an intermediate buffer between the CPU and the main memory. Its objective is to reduce the CPU wait time during the main memory accesses. A small but fast memory serves as a cache. A portion of the main memory is stored in the cache memory in advance. The cache controller keeps track of the locations of the memory, which are mapped in cache memory. When the CPU is in need of an instruction or operand, it is supplied from the cache memory. Thus, the slow main memory emerges as a fast memory. If the required item is not in cache memory, then an access to the main memory is performed. The capacity of the cache memory is kept very small since it is expensive. If the cache memory is full, some existing items are replaced for creating space for a new entry. The item for removal from cache memory is chosen by special algorithms. The hit ratio is a fraction of the number of accesses which have faced 'cache hit' to the total number of access. Some computers use a two level or three level cache memory system.

There are three popular methods of cache memory mapping: Direct, Associative and Set associative mapping. When a main memory block has to be stored, the cache line where the block has to be written is determined from the mapping function. Similarly, when the CPU reads from the main memory location, the cache controller, uses the mapping function to

identity the cache line where the main memory block is available. During memory write operation, two different strategies are followed: write-through policy and write-back policy.

The more frequent the cache hits are, the better it is since every 'miss' leads to accessing the main memory. The time taken to receive the required item from the main memory and supply it to processor is known as 'miss penalty'. Functionally, the cache memory has two parts: data memory and tag memory. Each line of the data memory contains a memory block. The corresponding tag line in the tag memory identifies the block number of the memory block.

There is a possibility of a main memory location getting updated without the knowledge of a processor. A typical example is an input operation from the hard disk when the DMA controller stores the data in the main memory. This causes mismatch between the contents of main memory and cache memory. To prevent this, a bus monitoring logic monitors the memory (bus) activities by the DMA controller (or other processors). This is known as snooping. On detecting the commencement of a memory write sequence for an address that has been mapped in cache memory, the cache controller is asked to invalidate the data in the cache memory. The cache controller sets INVALID flag for that entry.

In a virtual memory system, the operating system automatically manages the lengthy programs. The operating system stores the entire program on a hard disk. At a given time, only some portions of the program are moved into the main memory from the hard disk. As and when required, a portion which is currently not present in the main memory is taken from the hard disk and a portion of the program which is presently in the main memory is sent to the hard disk. This process is known as *swapping*.

In order to support the virtual memory, the program does not address the physical memory directly. While referring to an operand or instruction, it provides the logical address and the virtual memory hardware (memory management unit) translates it into the equivalent physical memory address.

There are two popular methods for the virtual memory implementation: Paging and Segmentation. In segmentation, the (machine language) programmer organizes the program into different segments of meaningful modules which may not be of same size. In paging, the system software divides the program into pages of same size. This is unknown to the programmer.

The associative memory is a special type of memory known as content addressable memory. In this, usually, a part of the content is used for accessing the content. The associative memory searches all the locations simultaneously in parallel. The associative memory is costly since it requires matching logic with each cell. Due to its high speed search capability, it is the best choice as cache memory and TLB in the address translation unit.

REVIEW QUESTIONS

1. A student of computer architecture concluded that the use of cache memory is unnecessary if a very large instruction queue and a large size write buffer are used inside the processor. Identify the flaws in this conclusion.
2. A processor has 4 KB of instruction cache and 4 KB of data cache both on-chip. The manufacturer finds a batch of 10000 processor ICs to be selectively defective. In 5000 ICs, the instruction cache is functionally missing. In the remaining ICs, the data cache is functionally missing. All the processor ICs have no other defect except for this problem. Throwing away the entire lot of 10000 ICs will cause a huge loss. Hence the top management asked the research department to provide a fallback or salvaging. Assume you are the head of the research department. The following suggestions are made by your team members:
 - (a) Sell all the ICs as processors without cache;
 - (b) Sell all the ICs as processors with 4 KB unified cache;
 - (c) Sell all the ICs to educational institutions and research labs at reduced price with clear instructions about the defect;
 - (d) Sell all the ICs as processors with 2 KB instruction cache and 2 KB data cache;
 - (e) Release the ICs announcing them as two new processor models:
Model A: With 4 KB instruction cache and no data cache
Model B: With 4 KB data cache and no instruction cache
 - (f) Piggy back two processors (one defective code cache chip with one defective instruction cache chip) and sell them as a single processor IC.
- Reject all bad (technically faulty) suggestions with valid reasons.
3. A computer with virtual memory uses hard disk drive to compensate for the limited physical capacity of the main memory. The penalty paid is increased instruction cycle time due to address translation process. The 'RAM disk driver' is a system program that emulates a hard disk drive using the main memory in order to reduce the number of accesses to the hard disk drive which is slow as compared to the main memory. The penalty paid is reduction in the usable main memory space for application programs. Don't you think that both the virtual memory and RAM disk driver feature cancel each other's benefit if both these features are available in a system.
4. Early computers were programmed in machine language. The programmer had the chance to deal with physical memory addresses. Once high level languages were introduced, the programmers lost their direct contact with physical memory addresses because of the availability of the compilers. The invention of virtual memory

took away this facility from the compilers. Hence, the programmers today are two levels away from the physical memory addresses. Does it mean that today the programmers have some disadvantage over the olden day machine language programmer? What is the impact on program execution time?

5. Intel 80386 processor's virtual memory system (address translation) supports both segmentation and paging. There is an option to disable paging and use only segmentation. If this option is exercised, the 80386's virtual memory system behaves similar to Intel 80286 processor as far as the virtual memory is concerned. When do you think this option should be chosen?

EXERCISES

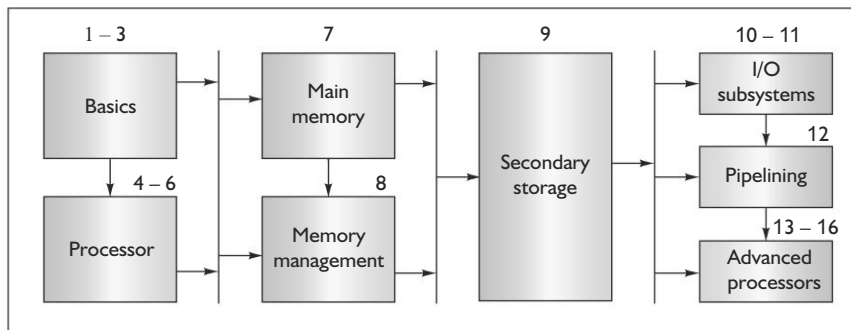
1. A computer's cache memory is 10 times faster than main memory. The hit ratio is 0.95.
 - (a) Calculate performance improvement in instruction fetch due to cache memory. Assume that the MISS results in moving the information from main memory to cache first and then from cache to CPU.
 - (b) It is observed that doubling the cache memory size halves the number of MISS cases. What is the change in performance improvement?
2. A computer's main memory consists of 1024 blocks of 256 words each. The cache memory has eight sets of each having four blocks. Determine the length of TAG, SET and WORD fields in the memory address.
3. A computer has 64 MB main memory and 32 KB cache memory. Each line in the cache stores 8 bytes.
 - (a) Show the format of the memory address for direct mapping, associative mapping and eight-way set associative mapping.
 - (b) How many lines are there in the cache?
4. A computer's cache memory follows write-through policy. The access time of the main memory is 100 ns whereas that of cache memory is 20 ns. The hit ratio for read access is 0.9. One-fourth of the memory cycles are for write and the rest for the read operations. Calculate the following:
 - (a) Hit ratio including both read and write cycles.
 - (b) Effective access time of the memory.
5. A memory has an access time of 20 ns. Calculate performance gain due to instruction prefetch assuming that 10% of the instructions are branch instructions.

6. A CPU has 1 MB memory space. Design a four-way interleaving with the memory ICs of 20 ns cycle time. Calculate the effective bandwidth.
7. A memory has an access time of 50 ns. Calculate performance gain due to write buffer if memory cycles for read and write are in the ratio of 3 : 1.
8. A computer has 24 bit logical address and 12 bit physical memory address. If the page size is 2 K, calculate the number of pages and the number of main memory blocks.
9. A computer's logical address space has 64 segments. The segment size is 64 K words. The physical memory has 1 K pages of each 4 K words. Determine the logical and physical address formats.

CHAPTER 9



SECONDARY STORAGE DEVICES



Chapter Flow Diagram

OBJECTIVES

- « Hard disk drive
- « Floppy disk drive
- « CD-ROM
- « DVD
- « Magnetic tape drive
- « RAID

KEYWORDS

Modified Frequency Modulation (MFM)	CD ROM
R/W Head	CD-R
Spindle Motor	CD-RW
Index sensor	RAID
Soft sector	Blade server
Hard sector	NAS
Winchester Hard disk	Hot spares
Rotational delay	Hot swapping
Seek time	Data striping
Track	Disk mirroring
Sector	Disk duplexing
Cylinder	Bit interleaving
Platter	Block interleaving
Diskette	Software RAID
Sector interleaving	Hybrid RAID

9.1 INTRODUCTION

The secondary storage devices play important role in a computer system. They help in reducing the cost of the computer system since their cost per bit is cheaper compared to main memory. A large secondary storage can serve as virtual memory thereby minimizing the capacity of main memory. In addition, the secondary storage is used for archiving programs and data. The need for a large capacity secondary storage is increasingly felt due to various new usages such as email, digital imaging, and digital audio and digital video.

Several types of secondary storage have been introduced in various types of computer systems from desktop PC to enterprise server. Magnetic storage devices and optical storage devices are two major types of secondary storage. This chapter covers the principles and usage of Magnetic Hard disk, Magnetic Floppy disk, Magnetic tape and Optical disk.

9.2 Magnetic Storage Devices

Figure 9.1 shows different types of magnetic storage devices used as secondary storage. The magnetic disk drive has been used since 1956. The invention of disk drives has made the magnetic drum obsolete. It has also reduced the use of the magnetic tape drive. Earlier, the tape drive was used as a resident unit for the operating system. The operating system is installed on disk drives, nowadays, because of their random access, providing faster access to data. The tape drive is used nowadays mainly for two purposes:

1. As a *back-up* unit, to take copies of the disk contents. These are useful, in case the disk contents are destroyed due to some problem.
2. To transport files from one site to another site, the tape media is more convenient.

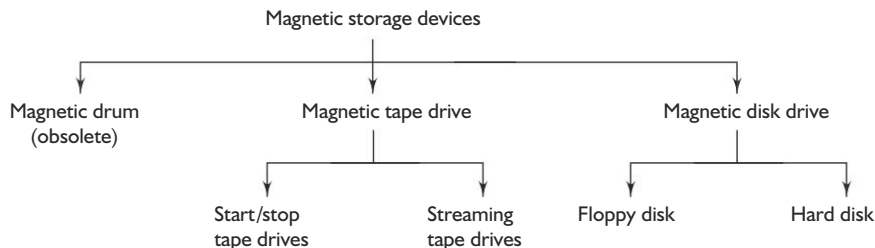


Fig. 9.1 Magnetic storage devices

The arrival of Compact Disc based on laser technology has changed the scenario. Large capacity, high reliability and easy portability are the plus points of this optical medium over the magnetic tape (and floppy diskette). Also the invention of flash ROM offers a replacement to hard disk in portable computers.



9.3 Basic Principle of Magnetic Disk

The basic principle of writing data on the disk and reading from the disk is same as that of tape drive. In both the devices, there is a magnetic medium. Data is stored on the magnetic medium by causing magnetisation of particles on the media. The magnetisation is caused by passing current through a coil in the read/write head. In both disk and tape, the head is stationary during the read/write operation and the media moves.

9.3.1 Recording Data on Magnetic Disks

Both analog recording and digital recording have been used. Analog recording was very popular in the past for audio and video recording. However, this has been gradually replaced by digital recording. In digital recording, only two stable magnetic states are present. Examples of digital recording are floppy disk and Hard disk.

There are two types of disk mediums: flexible diskettes and hard disks. Digital data is stored on a magnetic disk using magnetic pulses. These pulses are generated by passing a frequency modulated (FM) current in the magnetic head. This current generates a *magnetic field* that magnetizes the particles of the surface under the magnetic head. The pulse can be one of two polarities: positive or negative. Instead of storing digital data directly, it is encoded. Three popular encoding methods are (1) frequency modulation (FM), (2) modified frequency modulation (MFM), and (3) run length limited (RLL).

Older recording technique is based on Longitudinal recording whereas the modern technique used in hard disk is based on Perpendicular Magnetic recording (PMR). The difference is the magnetic orientation of data bits on the physical surface. In longitudinal recording, the magnetic orientation of data bits is aligned horizontally parallel to the surface of the disk. In PMR, data bits are aligned perpendicular to the disk surface. Hence, more small crystalline grains are fitted in the same surface area.

9.3.2 FM and MFM Formats

The FM format is known as single density format whereas the MFM method is known as double density format.

In the FM format, a clock pulse is written at the beginning of each *bit cell*. The data pulse is written at the centre of the bit cell. If the data is 1, the data pulse is present. If the data is 0, there is no data pulse. Each bit cell is of $4\ \mu\text{s}$ duration for floppy disk. Figure 9.2 shows FM recording format.

In the MFM format, the clock pulse is not present at the beginning of every bit cell. When the data is 1, there is no clock pulse. Only the data pulse is present at the centre of the bit cell. When the data is 0, following a 1 in the previous bit cell, neither clock pulse nor data pulse is written. But if the data is 0 both in the current bit cell and in the previous bit cell, then the clock pulse is written at the beginning of the current bit cell, but no data pulse is written in the bit cell. Figure 9.3 shows the MFM recording format.

In the FM recording, there are two flux changes per bit cell when 1's are recorded in all bit cells. In the MFM recording, since the clock pulses are eliminated, there is only one flux change per bit cell, when 1's are recorded in all bit cells. Hence, the duration of the bit cell in MFM is reduced to $2\ \mu\text{s}$ and the disk capacity is doubled in MFM.

When data is recorded on a magnetic medium, the information is stored in flux reversals on the medium and not in the amplitude or direction of magnetisation. Each data bit is recorded in the form of a flux change.

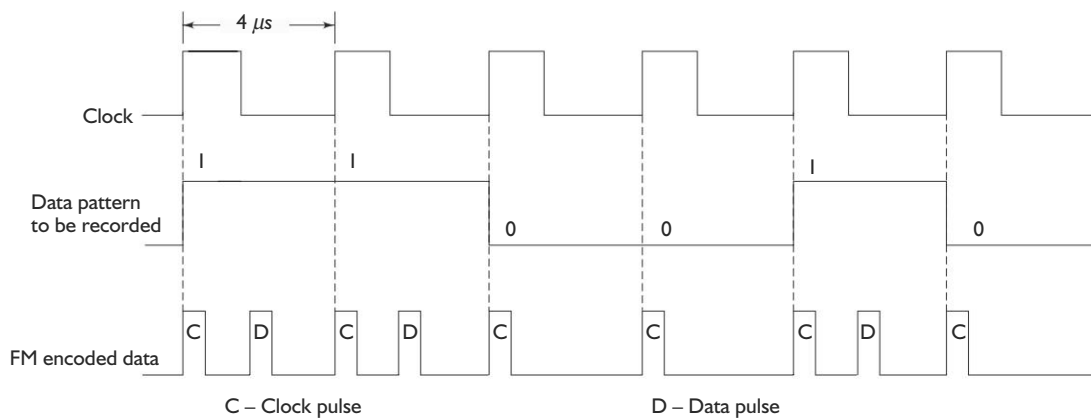
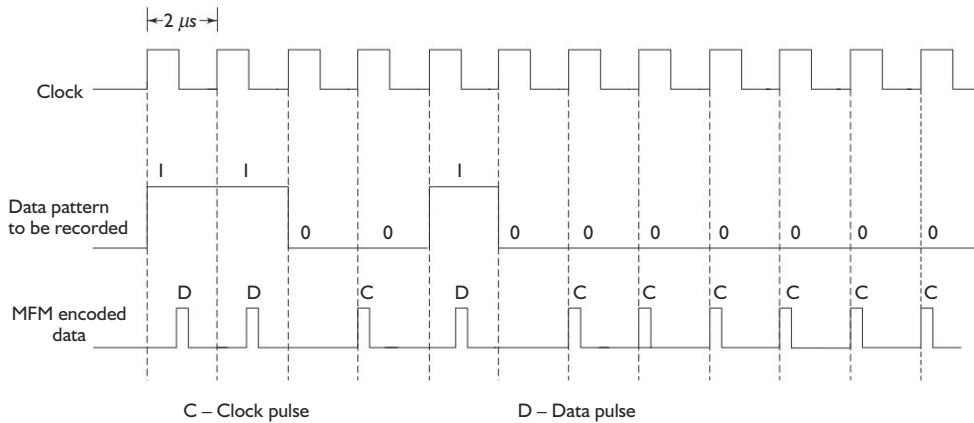
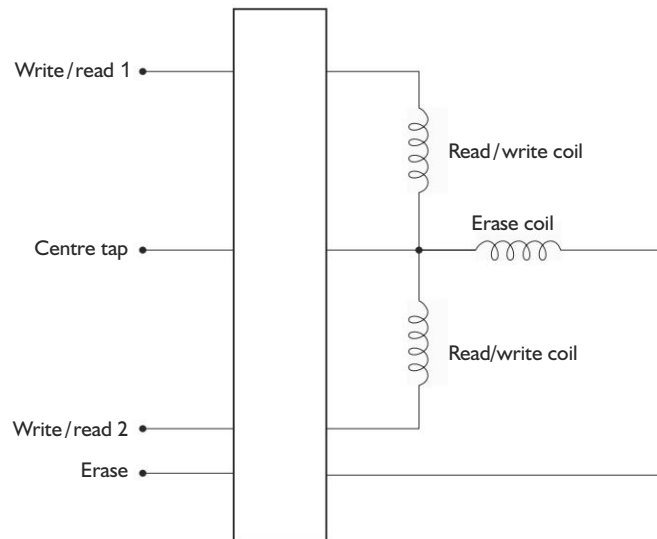


Fig.9.2 FM recording format

**Fig.9.3** MFM recording format

The read/write head consists of a core, with an air gap and a set of coils. Figure 9.4 shows the head configuration in a FDD. For writing data onto the medium, the data is converted into current and is passed through the read/write head coils. This current generates a magnetic field (flux) in the air gap. The current direction is controlled (alternately) to produce opposite polarity magnetic fields at the air gap. The result is—a series of *flux reversals* on the medium.

During reading, when the flux transitions pass under the head gap, a voltage is introduced in the read/write coils. This voltage is converted into data pulses.

**Fig. 9.4** Head configuration

In Fig. 9.5, the magnetisation due to the current in the head, during write operation, and the induced emf in the head, during read operation, are shown pictorially. FM recording is considered in this example.

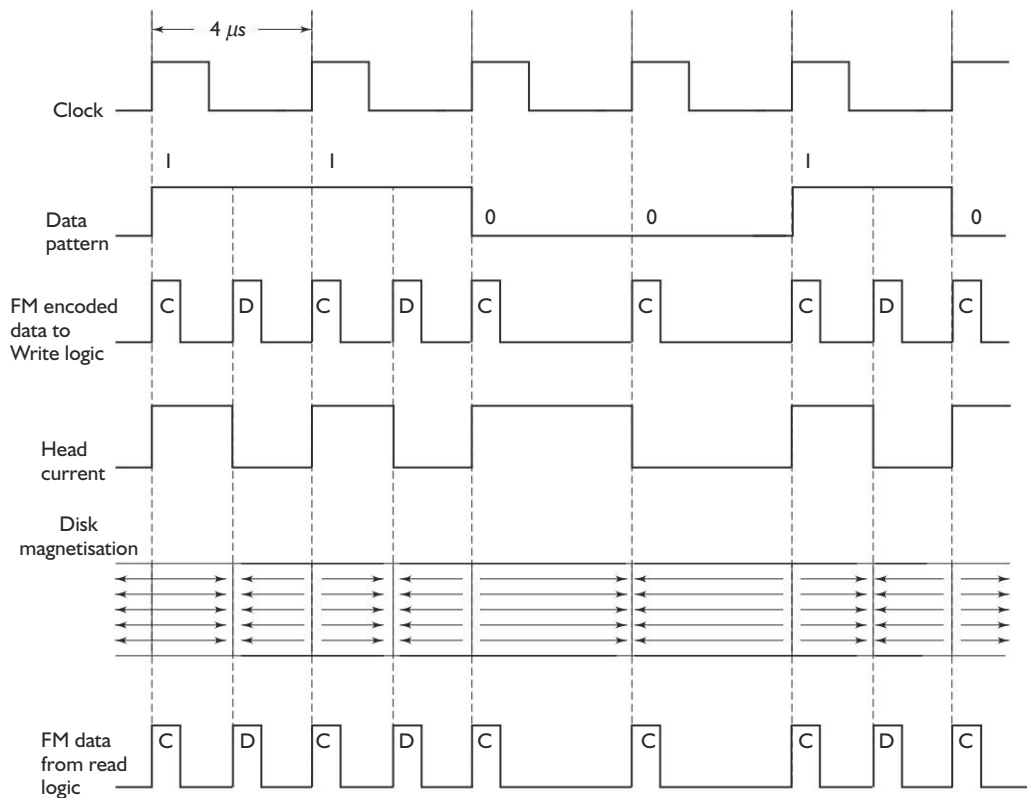


Fig. 9.5 Magnetic recording principle

9.3.3 Run Length-Limited (RLL) Encoding

The MFM method stores a “1” by generating a *no-pulse* (in the clock position), and a pulse in the middle. It stores a “0” as a *pulse* (in the clock position) and a *no-pulse* if the last bit was a “0.”. Instead it is stored as two no-pulses if the last bit was a “1.”

The RLL method is an improvement over the MFM encoding method. It reduces the run length (distance) between pulses (flux reversals) on the disk. The objective is storing more data in less space by reducing the number of flux reversals. There are many versions of the

RLL encoding. A popular version is the 2,7 RLL. This ensures that not less than 2 no-pulses and not more than 7 *no-pulses* can occur between pulses.

9.3.4 Disk Drive Types

Figure 9.6 shows the block diagram for the floppy disk drive. In a hard disk drive, the medium is a rigid circular platter, known as disk. In a FDD, the medium is a flexible circular one known as a diskette. In both these drives, two surfaces, top and bottom, can be used for storing data.

The HDD is more suited as secondary storage because of following reasons:

1. Higher capacity of data storage
2. Faster access time of data
3. Higher data transfer rate
4. Better reliability of operation
5. Less data errors or data loss

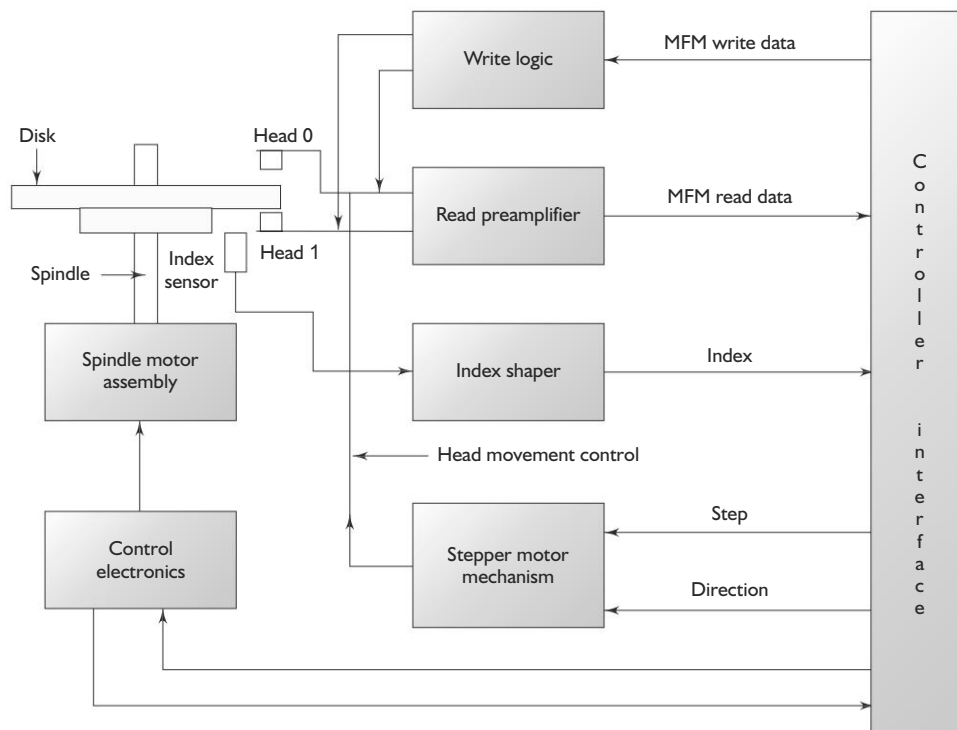


Fig. 9.6 Disk drive block diagram

Some of the common concepts between FDD and HDD are listed below:

1. Data is written bit by bit on the disk.
2. In addition to data bit, clock bit is also written on the medium.
3. Data is recorded on concentric circular tracks.
4. The disk rotates at a fixed speed.
5. The head is a moving head, which is moved to the desired track by a positioning mechanism.

The conceptual differences between FDDs and HDDs are listed below:

1. The head in the floppy disk touches the media surface during a read/write operation. In the HDD, the read/write head does not touch the media surface during a read/write operation. It *flies* above the disk at a minute distance, called flying height.
2. The FDD has two read/write heads one for each surface. The hard disk has many platters mounted on a single spindle. Each platter has two surfaces, top and bottom. Thus, the total number of read/write heads in a HDD is equal to twice the number of platters.
3. The head positioning mechanism in the FDD uses a stepper motor. In the HDD, two options are available:
 - (a) Stepper motor mechanism in low cost HDDs
 - (b) Voice coil servo mechanism in high performance HDDs
4. The diskette in the FDD is rotated at low speed, usually at 300 rpm or 360 rpm. The platters in the HDD are rotated at higher speed, nearly at 7200 rpm or 10,000 rpm.
5. The number of tracks on a floppy disk is lesser mostly 80. The track density is usually 96 TPI (Tracks Per Inch). In a HDD, higher track density is possible, and even 1000 TPI is common.
6. Since the hard disk rotates at a faster speed than the floppy diskette, the recording density, BPI (Bits Per Inch) in a HDD, is higher than the recording density in a FDD.



9.4 Floppy Diskette

A floppy diskette (Fig. 9.7) is an ultra thin plastic (Mylar) piece in circular shape. The thickness of the mylar disk is only a few thousands of an inch. It is coated with a magnetic material and enclosed in a protective jacket. An oval access hole (window) is made on the jacket so as to provide contact between the read/write head and the diskette. The diskettes are of different sizes as shown below:

1. 8 inch disk (8 inches square); this is an old standard diskette. Presently, it has become obsolete.
2. 5 1/4 inch disk (5 1/4 inches square); this diskette is known as mini-floppy. It was widely used in PCs.
3. 3 1/2 inch disk (3 1/2 inches square); this diskette is a new industry standard type. It is known as *micro-floppy*.

In the earlier floppy diskettes, only one side of the diskette was used to store information. Such a diskette is known as a single sided disk. In the current diskettes, both sides of the diskette are used for storing information. Such diskettes are known as double sided diskettes. The 5 1/4 inch disk has a vinyl cover whereas the 3 1/2 inch disk has a rigid plastic cover with a metal or plastic slide that protects the disk surface when the disk is not in use.

Depending on the recording technique used to store information, diskettes are classified into single density diskettes and double density diskettes. In double density diskettes, an improved recording technique (MFM) is used to store information. Due to this, a double density diskette stores twice the amount of information that can be stored on a single density diskette of the same size.

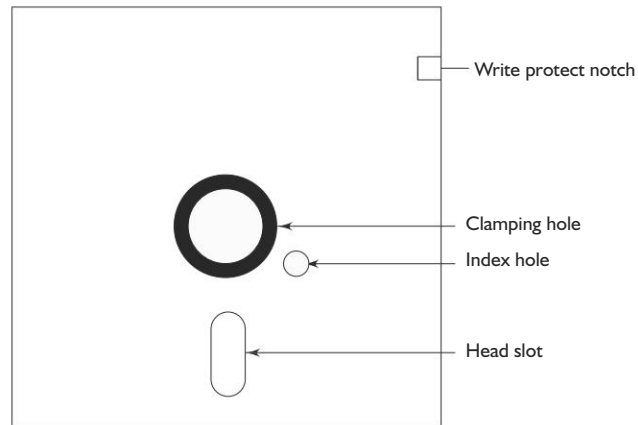
The diskette surface is logically divided into a fixed number of tracks or concentric circles (Fig. 9.7). The number of tracks in one surface of a diskette are 77, 40 and 80 for 8 inch, 5 1/4 inch double density diskette and 5 1/4 inch high density diskette respectively. Reading or writing takes place only on the specified tracks and not in-between the tracks.

The read/write heads are mounted on a common assembly in the FDD. The head assembly moves to and fro in steps between the outermost track and the innermost track. It can move in both directions: forward (towards the centre) and backward (towards the outermost edge). The outermost track is numbered as *track 0*. The subsequent tracks are numbered sequentially.

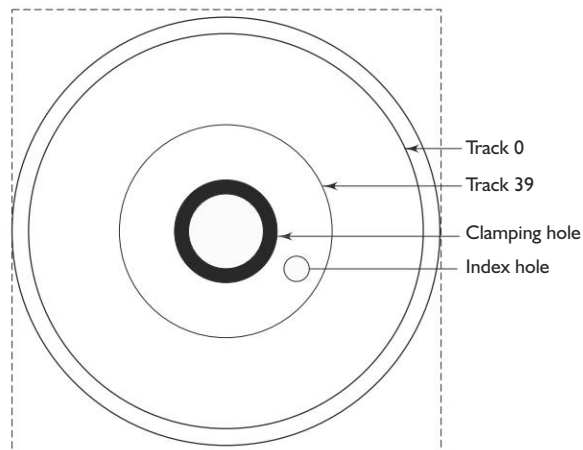
There is a small hole punched on the diskette near the centre. This hole is known as the index hole. It is a reference point indicating the beginning of a track. When the diskette rotates, the index sensor in the FDD senses the passing of the index hole. Initial writing on any track is done after the index hole is sensed. In other words, the beginning of any track is immediately after the index hole. When the diskette is rotating, the index sensor senses the index hole once on each revolution of the diskette.

Each track is divided into a number of sectors. The number of sectors in a track depends on the size and the recording method used. In each sector, a fixed number of data bytes are written. This number is generally one of the following: 128, 256, 512 or 1024.

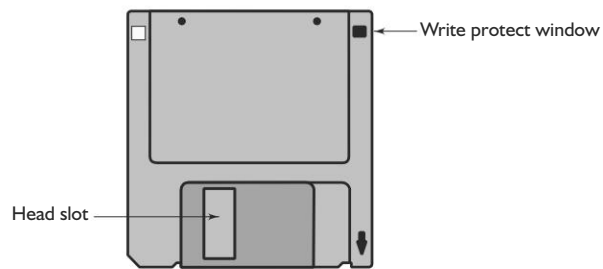
Figures 9.7(a) to (c) show both 5 1/4" and 3 1/2" diskettes.



(a) Floppy diskette in a protective jacket – 5¼ inch



(b) Tracks on a surface – 5¼ inch



(c) 3½" floppy disk inside plastic cover

Fig. 9.7 Floppy diskette

9.4.1 Write Protect Feature

Some time, the contents of a diskette may be overwritten by mistake. There is a facility to use a diskette only for reading previously stored information, and prevent any attempt to write new information on it. For this purpose, two different techniques are followed. In a 5 1/4" disk, there is a small notch punched at the outer edge of the jacket. This notch is known as the write protect notch. If this notch is open (uncovered), writing on the diskette is permitted. When this notch is covered by an insulated sticker, writing on it is not allowed. In a 3 1/2" disk, there is a write-protect window with a plastic tab which can be moved to open or close the window. When the window is open, the write-protection is provided.

9.4.2 Hard Sector and Soft Sector Formats

Depending on the sector organisation, the floppy diskette is classified into two types:

1. Hard sectored floppy diskette.
2. Soft sectored floppy diskette.

9.4.2.1 Hard Sectoring

In this method, the number of sectors on each track is physically fixed while manufacturing the diskette. The beginning of every sector is identified by a sector hole punched on the plastic disk. The hard sectoring is an obsolete concept.

9.4.2.2 Soft Sectoring

In this method, the number of sectors per track is fixed by the software. There are no physical holes on the diskette for sector information.

9.4.2.3 Hard Sector Versus Soft Sector Disk

In a hard sector floppy disk, the sector size is fixed. Hence, there is no flexibility. In a soft sectored floppy disk, the sector size is chosen by the software. Hence, different system software can select different sector sizes.

The hard sectoring (followed only in 8 inch floppy disks) is an obsolete concept and all newer systems use soft sectoring.

9.4.3 Floppy Disk Format

There are several standards for soft sectored diskettes. They differ in the formatting parameters like the number of sectors per track, number of data bytes per sector and the lengths of various gaps on the diskette. Figure 9.8 shows the basic concept of a soft sector format. Though many different standard formats have been adopted by different system manufacturers, the IBM 3740 format is the most widely used.

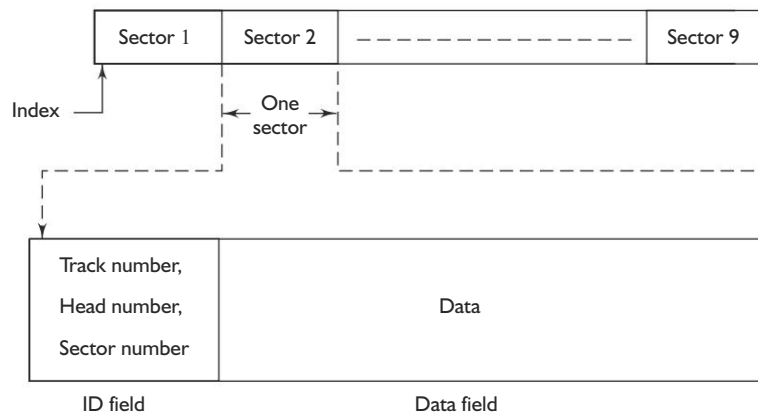


Fig. 9.8 Soft sector concept

IBM system-34 format is another standard format. Each track is formatted into many fields, containing specific bit patterns and gaps. The bit patterns in the fields, other than the data field, are used for synchronisation, error detection and identification. The gaps are *no-data* regions.

9.4.3.1 IBM Compatible Floppy Disk Format

Each track is divided into a number of sectors. In a PC, the older operating systems such as DOS organised 9 sectors whereas the current standard is either 18 or 36 sectors on a track with 512 bytes of data on each sector. Each sector consists of an ID field and a data field. The ID field contains the address of the sector. The data field contains the actual data stored in the sector. The ID field is written in each sector of a track while the track is formatted. Formatting a diskette erases any previous data stored in it and writes the sector ID fields and certain other information necessary for synchronisation. Formatting also writes a

specific data pattern in the data fields of all the sectors. Thus, a new diskette, after formatting, is not a blank diskette.

The standard floppy diskette format used by IBM double density recording format consists of pre-index gap, address mark, post index gap, sectors and final gap. The different fields are illustrated in Fig. 9.9.

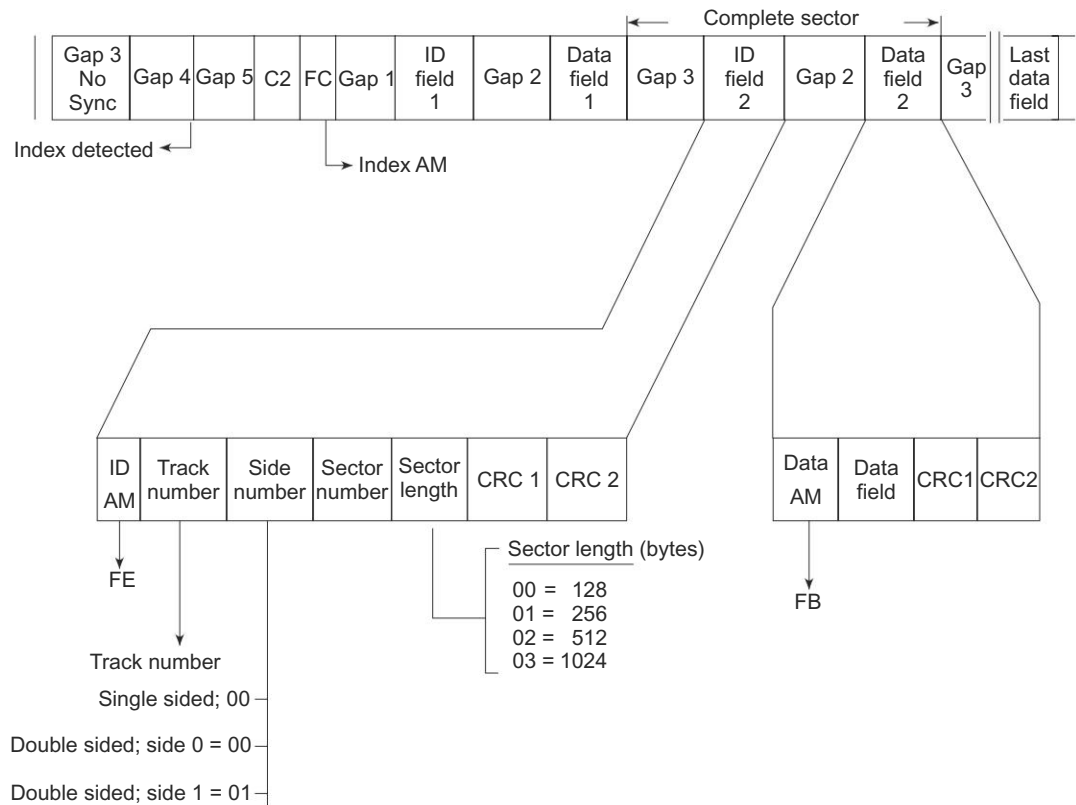


Fig. 9.9 IBM compatible sector/track format

Address marks are unique bit patterns used to identify the beginning of ID and Data fields. Certain bit cells in the address mark bytes do not contain a clock bit. There are four different types of address marks to identify the different types of fields. They are: Index address mark, ID address mark, Data address mark, and Deleted data address mark. Table 9.1 defines all the four types of address marks.

The ID field is seven bytes long consisting ID address mark, Cylinder number, Head number, Record number (Sector number), Sector length, and CRC bytes. The bad track is identified by writing hexa FF in the ID fields of the bad track. The gaps separating the different fields are created on the track when the diskette is formatted. These gaps serve multiple purposes, as listed below:

1. Providing Time interval for the FDD circuit to select the required mode: read/write
2. Compensation for variations in diskette rotational speed
3. Compensation for tolerances in diskette and FDD so as to maintain compatibility

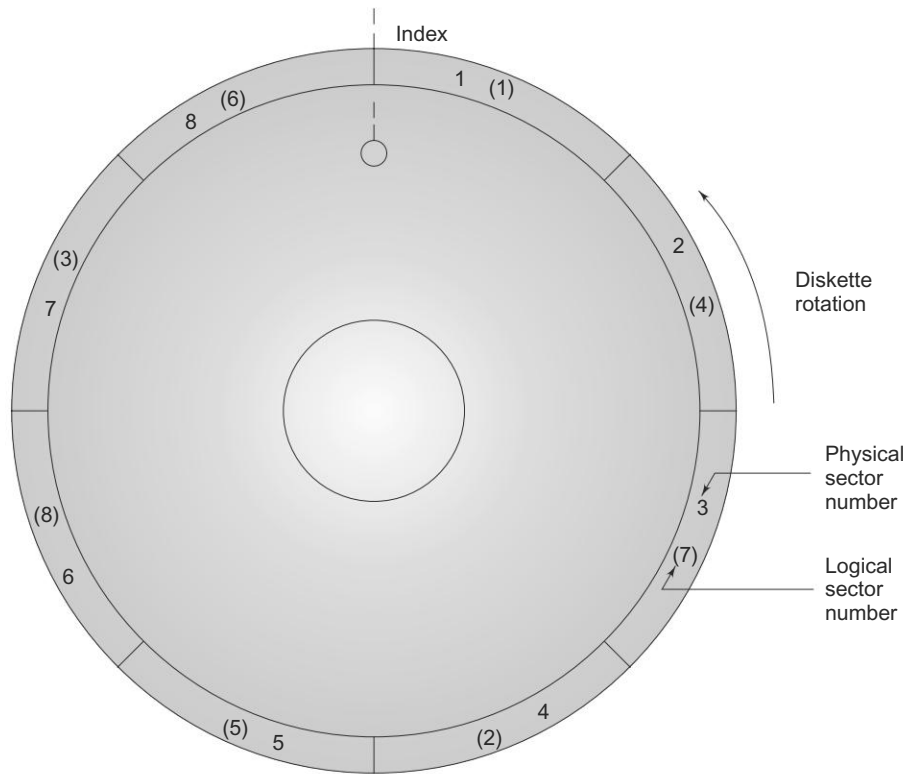
TABLE 9.1 Types of address marks

Sl.No.	Address mark	Description	Clock pattern	Data pattern
1	Index Address Mark	Located at the beginning of each track	D7	FC
2	ID Address Mark	Located at the beginning of each ID field	C7	FE
3	Data Address Mark	Located at the beginning of each non-deleted data field	C7	FB
4	Deleted Data Address Mark	Located at the beginning of each deleted data field	C7	F8

9.4.4 Sector Interleaving

While formatting a diskette, the sector number is written in the ID field of each sector. It may appear that the sectors are numbered sequentially in the physical order of presence. Though this is possible, there is a better technique of sector numbering, known as sector interleaving, wherein the sectors are numbered in a non-consecutive order. Suppose in this method, the physically first sector is numbered as sector 1, and the physically fourth sector is numbered as sector 2. We have three sectors difference between the logical sector 1 and the logical sector 2. Then, the logical sector 3 is the physically seventh sector. Such an interleaving, is said to have an interleave factor of 3:1. Figure 9.10 shows a track organisation with 3:1 interleaving. The format procedure establishes the interleave factor by writing the logical sector numbers in the ID fields of each sector. Once it is formatted with the sector interleaving factor, the disk controller recognises only logical sector organisation.

The objective of sector interleaving is to provide an efficient way of files organisation with optimum access time of records and minimum idle time. The exact interleave factor should be decided by considering the CPU speed and the diskette rotation speed.



Note: Logical sector numbers are indicated within brackets.

Fig. 9.10 Sector interleaving concept



9.5 Overall Operation of Floppy Disk Subsystem

The I/O operations related to the FDD are done by coordination between various software and hardware modules: BIOS routines, OS, application program or user command to OS, DMA controller, FDC and FDD. Figure 9.11 presents an overview of floppy disk subsystem.

Floppy Disk Drive Functions

The floppy disk drive performs the following operations:

1. Loading the R/W Heads
2. Moving the R/W Heads by one track for every STEP pulse. The DIRECTION signal decides the direction of the R/W Heads Movement

3. Writing data (data, clock, gap, sync, etc.) on the diskette, when the WRITE ENABLE signal is active
4. Reading data (data, clock, gap, sync, etc.) from the diskette, when the WRITE ENABLE signal is inactive
5. Rotating the diskette when MOTOR-ON signal is active
6. Responding to the signals from the controller when DRIVE SELECT is active
7. Presenting the outputs of Index sensor, Write Protect sensor, Track 0 sensor, etc., to the controller when DRIVE SELECT is active

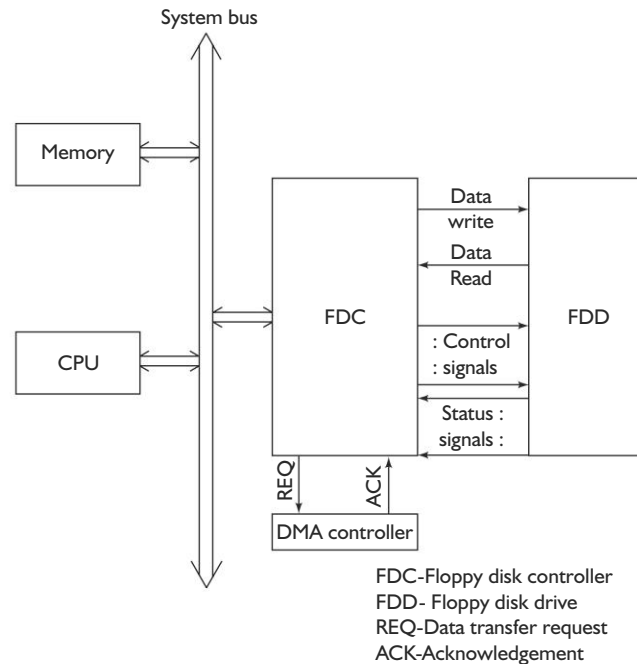


Fig. 9.11 Overview of floppy disk subsystem

Floppy Disk Controller Functions

The FDC is an intelligent subsystem in the PC. Its main functions are summarised below:

1. Analysing the command received from the CPU, executing the command, and building up various status parameters indicating the completion of the command.
2. Receiving the data byte from the system data bus; serialising the data byte into bit streams; adding clock bits with the data bits as per the MFM format.
3. Separating clock bits and data bits, from the bit stream in the diskette; deserialising the data bits into data bytes and presenting them to the system.

4. Generating the CRCC while writing on the diskette.
5. Recalculating the CRCC while reading from the diskette and verifying it.
6. Locating the desired sector by matching the sector ID with the desired sector number.
7. Maintaining the current track number (position of the R/W heads).
8. Determining the desired direction of movement of R/W heads and issuing DIRECTION signal to the FDD.
9. Calculating the difference between the current track number and the desired track number and issuing corresponding number of STEP pulses to the FDD.
10. Maintaining the status of the FDD.
11. Generating DMA request for every byte of data transfer.
12. Generating interrupt at the end of a command.
13. Issuing STEP pulses and DIRECTION signal to the FDD to bring the read/write heads to track 0 during the RECALIBRATE command.
14. Forming different status bits to indicate various error conditions.
15. Creating necessary bit patterns and gaps on the diskette during FORMAT command.

Software Functions

In the PC, the BIOS ROM in the motherboard contains fundamental software routines required for performing various sequences of operations on the floppy diskette. Each routine is *called* by the operating system or utility program by supplying certain parameters, such as track number, head number, sector number, etc. For example, if the contents of one floppy diskette A have to be copied on another diskette B, the software (OS or utility) should perform the following sequence:

1. Read the contents from each track of A and transfer them to memory.
2. Get the memory contents and write on each track of B.

There are two methods for doing these two steps:

- (a) Read from A and write into B, track by track.
- (b) Read all the tracks from A, and then write all the tracks in B.

The reader must notice three important points:

1. There is no direct data path from one FDD to another FDD.
2. The FDC cannot send data from one FDD to another FDD.
3. Any read operation involves data transfer from the FDD to memory via the FDC. Similarly, write operation involves data transfer from memory to FDD via the FDC.

The user gives the COPY A, B command to the operating system, DOS. The DOS, in turn, requests the BIOS repeatedly to perform *reading* from A and *writing* on B. The DOS supplies the required parameters such as drive number (A or B), track number and the operation desired (read or write), whenever it calls the BIOS. On receiving this information, the BIOS, in turn, issues the command and parameters to the FDC and DMA parameters to the DMA controller.

9.5.1 FDD Specifications

Current PCs use the 3 1/2 inch FDD known as microfloppy drive. Two types of 3 1/2 inch FDDs are currently being used: double density FDD and high density FDD. The 3 1/2 inch double density FDD has 720 KB storage capacity and the high density has 1.4 MB storage capacity. The latest 3 1/2 inch FDDs offer 2.88 MB storage capacity. The number of sectors in the 720 KB 1.4 MB and 2.88 MB FDDs are 9, 18 and 36 respectively. Table 9.2 compares the different FDDs.

TABLE 9.2 Different FDD models

S. No.	Item/feature	5 1/4 inch; 360 KB	5 1/4 inch; 1.2 MB	3 1/2 inch; 720 KB	3 1/2 inch; 1.44 MB	3 1/2 inch; 2.88 MB
1	Capacity (formatted)	360 kB	1.2 MB	720 kB	1.44 MB	2.88 MB
2	Tracks/side	40	80	80	80	80
3	Sectors/track	9	15	9	18	36
4	Bytes/sector	512	512	512	512	512
5	Data transfer rate (kbs)	250	500	500	500	500
6	Recording format	MFM	MFM	MFM	MFM	MFM
7	Disk speed (rpm)	300	300 or 360	360	360	360
8	Total sectors	708	2371	1426	2847	5726
9	Common terminology	DSDD	DSHD	DSDD	DSHD	DSQD

DSDD—Double-sided double density; DSHD—Double-sided high density; DSQD—Double-sided quad density; b—bit; B—byte.

9.5.2 Head Movement

One read/write head is present for each surface. Both the heads are mounted on a common assembly; hence, both heads always move together.

The heads are moved forward (in) or backward (out) by a head actuator mechanism. This consists of a stepper motor and a metal belt. The stepper motor rotates in discrete steps and can rotate in both directions. It is coupled to the head assembly by the metal belt which is a coiled split steel band. The rotational movement of the stepper motor is converted into the linear movement of the heads by the steel band, which winds or unwinds around the spindle of the stepper motor.

The heads are spring loaded. This is done by the head load mechanism which consists of a solenoid. The head load time is generally of the order of 50 ms.

9.5.3 Head Coils

The read/write head consists of two R/W coils and one erase coil, to perform the read, write and erase functions.

The erase winding is energised, during writing, to reduce the width of the track. The erase coil does not erase the track but only trims it to a desired width, thereby creating guard bands (or gaps) between tracks.

There are two types of erase techniques: the *straddle erase* and the *tunnel erase*. In the tunnel erase head, the erase operation is done after the write operation. In the straddle erase head, both are done simultaneously.

9.5.4 Spindle Motor

The spindle motor rotates the diskette at 360 rpm. There are two types of spindle motor assemblies used in FDDs: belt driven and direct drive. The earlier FDDs used the belt driven spindle motor assemblies. In this, a belt and a pulley are used to couple the spindle motor to the disk spindle. The modern FDDs use a spindle motor whose shaft drives the head assembly directly. This type of spindle motor assembly is smaller and more reliable. They also have a tachometer control circuit to maintain a constant speed.



9.6 Hard Disk Drive (HDD)

The HDD has been existing for more than 50 years, since the introduction in 1956 by IBM. It is used as a secondary storage for storing data or programs.

In the HDD, there are multiple platters (or disks) mounted on a common spindle (Fig. 9.12). Each platter has two magnetic surfaces, top and bottom. The platter is made from a non-magnetic material, such as aluminum alloy or glass. It is coated with a thin layer

(10-20 nm) of magnetic material. In addition, an outer layer of carbon is coated for protection. Early disks had iron oxide as the magnetic material, and later disks used a cobalt-based alloy. The present trend is using thin film disks.

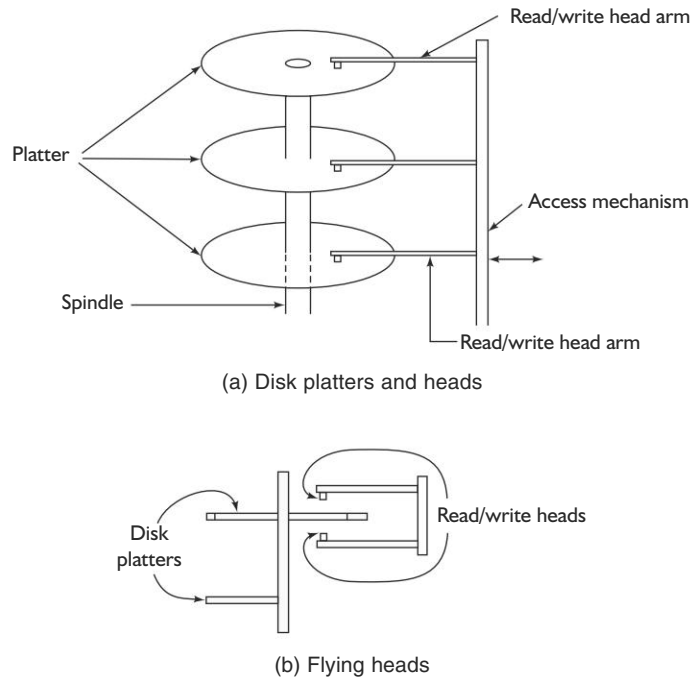


Fig. 9.12 Multiple platters on common spindle

Older disks followed fixed BPI/TPI formatting. The aerial density achieved is 40 Gb/sq. in maximum. Currently, adaptive formatting technique is used to achieve 70 Gb/sq.in. In adaptive formatting, the media is formatted in the factory, with optimised Bits per inch/Tracks per inch combination depending on the characteristics of the head with that surface.

The write head creates a magnetic region with a strong local magnetic field. Initial HDDs used inductive heads that generated this field by an electromagnet, that is also used to read the data by electromagnetic induction. The later versions of inductive heads are metal in Gap (MIG) heads, and thin film heads. These were followed by read heads using magnetoresistance (MR) that offered increased data density. In these, the electrical resistance of the head changed according to the strength of the magnetism from the platter. Later, “giant” magnetoresistance (GMR) heads using spintronics were introduced; in these heads, the

magnetoresistive effect was much higher. In today's hard disk drives, the read and write heads are separate, but in close proximity, on the actuator arm. The read head is magneto-resistive while the write head is thin-film inductive. In a thin film disk, the thin film surface is affixed onto the substrate. A read/write head flies over the disk and performs the read or write operation. The early HDDs used ferrite heads, whereas modern HDDs use thin film heads. As in FDD, data is recorded on the disk on concentric circles, called tracks. The tracks on a specific circle on all the surfaces put together form a cylinder. Hence the number of cylinders in a given HDD is the same as the number of tracks in a surface. Figure 9.13 illustrates tracks and cylinder.

The data from the computer comes in serial form. The serial bit stream includes both DATA bits and CLOCK bits. The read/write head records the data and clock information on the track. The control signals received from the computer inform the HDD of the exact location (cylinder number, surface number) where the data has to be recorded.

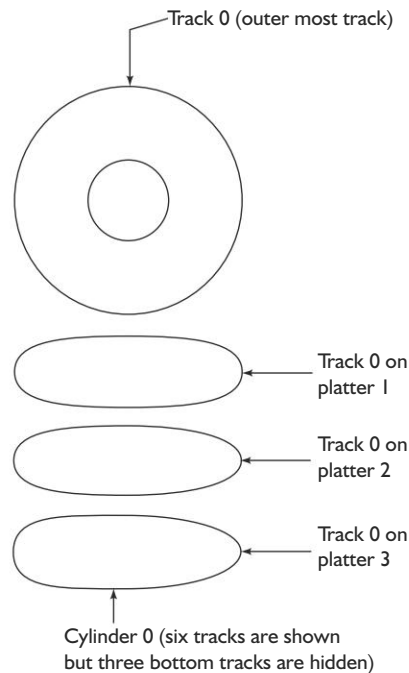


Fig. 9.13 Hard disk tracks and cylinders

While reading, the head reads everything (data and clock) and sends the bit stream to the computer. The disks rotate at high speeds: 7200 rpm, 10,000 rpm or 15,000 rpm.

The HDDs are of many types, as shown in Fig. 9.14.

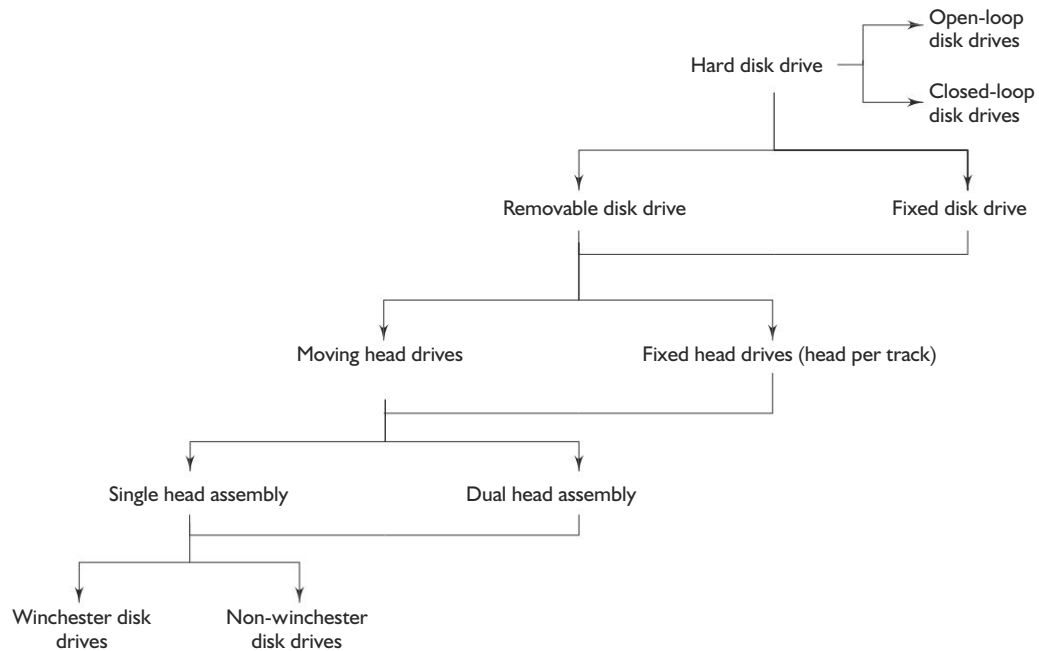


Fig. 9.14 Hard disk drive types

9.6.1 Removable Disk Drive and Fixed Disk Drive

In a removable disk drive, the disks can be removed and stored in cupboards. Also, data created in one disk drive can be transported to another drive/computer. In a fixed disk drive, the disk cannot be removed from the disk drive.

9.6.2 Moving Head and Fixed Head Disk Drive

In a moving head disk drive, the read/write heads move from one track to another track. The track selection is by mechanical movement whereas the surface selection is electronic. In a fixed head disk drive, the read/write heads are fixed and not movable. Hence, there

must be one head for each track. Both track selection and surface selection are electronic. This type is used in special purpose computers only.

9.6.3 Single Head Assembly and Dual Head Assembly

In a single head assembly drive, there is one head for each surface. These are mounted on one spindle and move together. At any instant, all the heads are positioned on the same cylinder.

In a dual head assembly drive, there are two heads for each surface. When first set of heads is in the home position, i.e. cylinder 0, the other set of heads is positioned over the middle number cylinder. When the first set comes to the middle number cylinder, the second set reaches the final cylinder. Thus, one set of heads cover the first half cylinders, and the second set of heads cover the second half cylinders.

9.6.4 Winchester and Non-Winchester Disk Drive

The term *Winchester* refers to a new HDD technology introduced by IBM in 1973. The Winchester technology has created a new revolution in HDDs.

9.6.4.1 Winchester Hard Disk

The name Winchester refers to the similarity of the first Winchester hard disk to the famous Winchester 30/30 rifle. The design of IBM's first Winchester hard disk was a dual 30 MB drive (two disks each of 30 MB in a single unit). This was referred to as 30-30. This notation brought the term Winchester to the new technology.

The features of Winchester technology are summarised below:

1. Read/write heads and disks are contained in a sealed enclosure.
2. The head flies very close to the hard disk, less than 19 microinches.
3. The heads park on the parking zone (landing zone) when the disk is not rotating. They take off and fly on a thin layer of air when the disk starts. No data is written in the landing zone.
4. The surface of the disk is lubricated to prevent damage to the head or track.

Higher track densities and bit densities are achieved in Winchester technology. Though there is a built-in preventive system against *head-crash*, i.e., the head touching the disk, any accidental head-crash destroys data stored in the disk. Hence, the user should take a back-up of data at frequent intervals.

9.6.4.2 Head Crash

Spring tension from the head mounting pushes the heads towards the platter. However, the heads have no physical contact or wear. The heads are prevented, from touching the platter surface, by the air that is nearer to the platter. The air moves almost at the platter speed. It is a type of air bearing inside the enclosure that supports the heads at their flying height while the disk rotates. The connection to the external environment is through a small hole in the enclosure, with an internal filter. If the air pressure is too low, the head gets too close to the disk, and there is a risk of head crash. The air inside the drive is constantly moving, being swept in motion by friction with the spinning platters. This air passes through the internal filter to remove any particles or chemicals.

Due to the minute spacing between the heads and the disk surface, any dust on the read-write heads or platters causes a head crash i.e. the head scratches the disk surface, (chipping away the magnetic film) resulting in data loss. Some common causes for the head crashes are electronic failure, sudden power failure, physical shock, wear and tear, or corrosion.

9.6.5 Open-Loop and Closed-Loop Disk Drive

In the open-loop system, a stepper motor is used for head movement, whereas voice coil positioner is used in the closed-loop system.

In the open-loop system, track 0 is the reference point. To move the head from the present track to the next track, a STEP pulse is issued to the stepper motor. The number of step pulses decide the total distance of head movement. There is no feedback to verify whether the head has really moved or not, and whether the head has moved to the correct track or not. Hence, the open-loop system is not *foolproof* and the software should read the track formatting information to ensure the head is positioned over the desired track. Figure 9.15 shows the signals involved in the open-loop system. The direction signal decides the direction of movement of the head: forward (towards the centre) or reverse (towards the zero track). When the head is positioned over track 0, the HDD sends TRACK 0 signal to the computer. The computer always keeps the present track number with reference to the track 0 position. The cost of an open-loop positioning system with a stepper motor is very low, and hence, is followed in low cost disk drives.

Figure 9.16 shows the principle of a closed-loop positioning system. In a closed-loop positioning system, the information stored on the hard disk is used as reference information for positioning. Head positioning information is stored on the hard disk in one of the following two ways: embedded servo technology and dedicated servo technology

In the first, on each surface, some tracks called servo tracks contain positioning information. These servo tracks are additional tracks other than data tracks.

In the second, a single surface, called 'servo surface' contains positioning information. The servo surface is dedicated for positioning information and does not contain any user data.

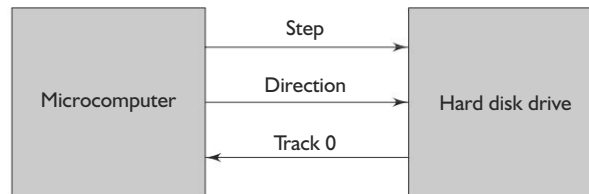


Fig. 9.15 Open-loop positioning system

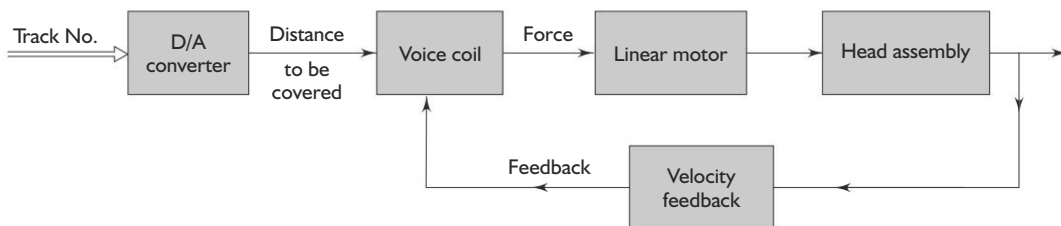


Fig. 9.16 Closed-loop positioning system

In a closed-loop system, the head gets positioning information from the disk. A linear voice coil mechanism is used in a closed-loop system. This mechanism controls the position of a linear motor using feedback control. It can move the head at different velocities. If the head has to move by a large distance, the head is accelerated at a high velocity. When the distance to be covered has reduced considerably, the mechanism decelerates the head, i.e. the velocity reduces. This process continues until the velocity becomes zero as the head has reached the desired track. If the head is positioned slightly off from the desired track, it is automatically repositioned.

Accurate head positioning over the desired track is possible in a closed-loop system. Also the time taken to position (seek) over a track is less. But the cost of the closed-loop system is higher than the open-loop system. It is suitable for high performance systems.

9.6.6 Size and Capacity

The HDD size is usually specified in inches. For example, an 8-inch HDD implies that the disk platter's diameter is 8 inches. Presently, the 2 ½ inch hard disks are widely used in present day microcomputers.

The capacity of an HDD can be calculated by multiplying the number of cylinders by the number of heads by the number of sectors by the number of bytes/sector (usually 512). The 3.5" and 2.5" hard disks currently dominate the market. One of the first 3.5" HDD to store 1 TB was the Hitachi Deskstar 7K1000. It contains five platters at approximately 200 GB each, providing 1 TB of usable space. In recent HDDs, a single 3.5" platter is able to hold 500GB worth of data. Table 9.3 gives the physical data of some typical HDD models that have been commercially available during the past 30 years.

TABLE 9.3 Typical hard disk sizes and capacity

Sl. no.	Form factor	Width	Capacity	No of platters
1	5.25" Full height	146 mm	47 GB	14
2	5.25" Half height	146 mm	19.3 GB	4
3	3.5"	102 mm	2 TB	5
4	2.5"	69.9 mm	1 TB	3
5	1.8"	54 mm	250 GB	3
6	1.3"	43 mm	40 GB	1
7	1"	42 mm	20 GB	1
8	0.85"	24 mm	8 GB	1

HDDs were initially used with general purpose computers. The need for large-scale, reliable storage, led to the introduction of RAIDs, network attached storage (NAS) systems, and storage area network (SAN) systems that provide efficient and reliable storage for large volumes of data. Recently, HDD usage entered the consumer applications such as camcorders, cellphones, digital audio players, digital video players, digital video recorders, personal digital assistants and video game consoles.

The capacity of the HDD depends on the size and technology. In 1956, a 24 inch HDD offered hardly 5 MB capacity. Currently, even a $2\frac{1}{2}$ inch HDD offers 1 TB capacity. The developments over the years provide higher track density, higher recording density and better packaging methods, offering more capacity.

9.6.7 Hard Disk Drive Organization

The Hard disk drive contains all the necessary mechanical and electronic units/sub-assemblies to analyse control signals, position the read/write heads on desired track, read and write data and establish a contaminant-free environment for the heads and disks. The functional units in a HDD are shown in Fig. 9.17a.

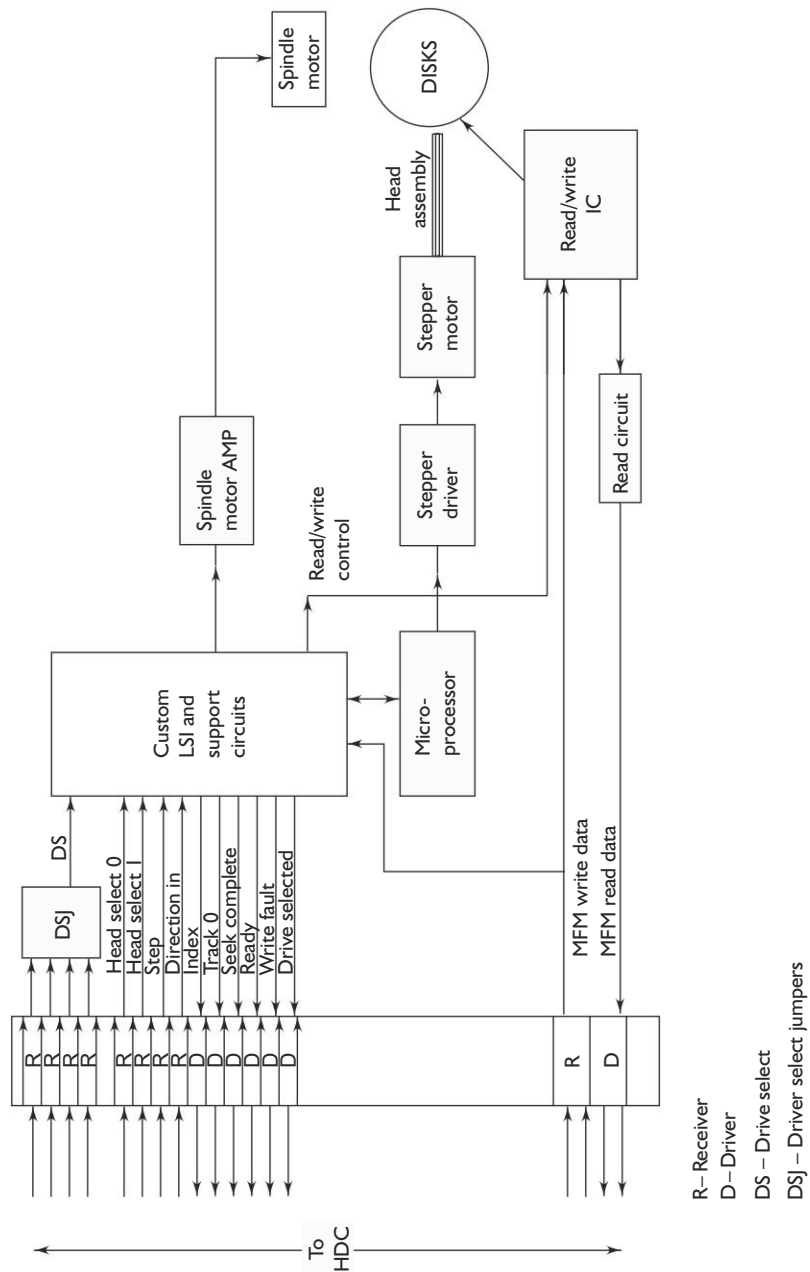


Fig. 9.17(a) HDD functional units (ST-506)

A hard disk drive consists of both electronic circuits and electromechanical subsystems (Fig. 9.17b). The subsystems in a hard disk are:

1. Read/write head Assembly
2. Disk platters
3. Spindle motor Assembly
4. Positioning mechanism
5. Air circulation system
6. Air filters
7. 'Track zero' detection sensor
8. 'Index' detection sensor
9. Read signal pre-amplifier
10. Write current drive
11. Interface logic

In a Winchester hard disk, the sealed Head Disk Assembly (HDA) consists of read/write heads, disks and band actuator assembly and air circulation system.



Fig. 9.17(b) Inside harddisk drive

The hard disk drive's electronics perform following operations:

1. Controls the movement of the actuator, and position the Read/Write heads
2. Maintains rotation of the disk at constant speed
3. Reads and writes on disk surface as requested by the disk controller.

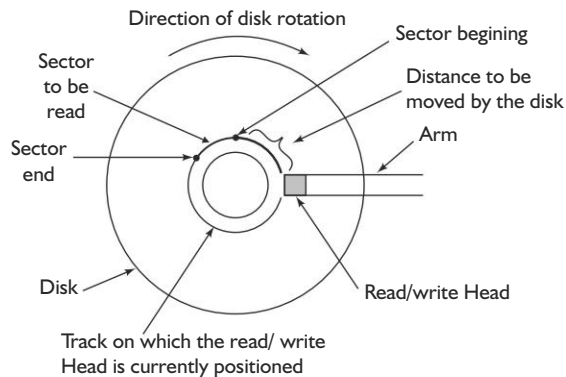
A hard disk drive has two motors: spindle motor and linear motor. The spindle motor is usually a brushless dc direct drive motor to rotate the disks. The linear motor (or the stepper motor) positions the read/write head assembly. The actuator arm moves the heads on an arc across the platters as they rotate. The air pressure inside the enclosure supports the heads at their flying height while the disk rotates.

9.6.7.1 Access Time

Average disk access time = Average seek time + Average rotational delay + transfer time + controller overhead

Reduction of access time is possible by increasing the rotational speed, thereby reducing rotational delay. The time taken to access a particular record consists of three parts:

1. Seek time: Time to move the heads to the desired track.
2. Rotational delay (latency): Time for the sector to come below the head (Fig.9.18).
3. Transfer time: Time to transfer the data. This depends on the sector size, the rotation speed, and the recording density of a track.



Note: After the Read/Head is positioned on the desired track, the desired sector's beginning has to be moved below the Read/write Head before the read operation can start. The rotational delay depends on the exact sector required.

Fig. 9.18 Rotational delay

The seek time and latency depend on the sector required at any time relative to the current position of the heads. Seek time, presently, ranges from 2 ms to 15 ms. The stepper motor drives had large access times (80–120 ms), but voice coil type drives have access times less than 20 ms.

The throughput and storage capacity depends on areal density. A typical 7200 rpm desktop hard drive has an average data transfer rate of about 70 megabytes per second. The controller overhead depends on the design organization of the hard disk controller.

9.6.7.2 Landing Zones

Power fluctuations or certain malfunctions can cause landing of the heads in the data zone. This is prevented by parking the heads either in a landing zone or by unloading (i.e., load/unload) the heads. The landing zone is usually near its inner diameter and no data is stored there. Either a spring or rotational inertia in the platters is used to park the heads during unexpected power loss. In such a case, the spindle motor temporarily acts as a generator, providing power to the actuator. In Load/Unload technique, the heads are lifted off the platters into a safe location.

Modern Drive Specifications

Performance improvements have been rapidly achieved over the years in various aspects of hard disk technologies. Table 9.4 provides the specifications of a modern disk drive. The transfer rate from the disk to the buffer is higher on outer tracks than inner tracks since there are more sectors on outer tracks compared to inner tracks.

TABLE 9.4 Hard disk drive specifications

Sl No.	Parameter / feature	Old hard disk drive	Modern hard disk drive
1	Capacity	30 MB	1 TB and above
2	Rotational speed	3600 rpm	7200 to 15000 rpm
3	Average seek time	45 msec	2 to 15 msec
4	Sustained transfer rate (Disk to buffer)	5 MB/s	70 to 145 MB/s
5	Host transfer rate	„	3 Gbps
6	Cache (buffer)	512 Bytes	32 MB

9.6.8 Data Organization on Hard Disk

On a hard disk, data is stored on tracks similar to Floppy disk. A track is subdivided into a number of sectors. Data is recorded on each sector. On each sector, the track number, head number and sector number are written in the ID field.

The capacity of a hard disk depends on the number of cylinders, number of surfaces, number of heads and recording density. In the soft sector format, the system software defines the sector format. The OS organizes the track into required number of sectors. The first sector on the track 0 is the master boot sector. It contains information regarding the *partitioning* of the hard disk. If this sector is physically damaged, the hard disk becomes

unusable. A special SYNC field and a set of GAP fields are created on the hard disk to satisfy the physical timings in the HDD. The following is the track format:

1. GAP 4 at the end of last sector in each track: it takes care of spindle speed variations
2. GAP 1 after the index: a cushion for the stabilization of the read amplifier while reading the first sector
3. All 0's of 14 bytes
4. ID field
5. All 0's of 15 bytes
6. Data field
7. All 0's of 3 bytes
8. GAP 3 at the end of each sector: a protection against overwriting on adjacent sector when creating a new sector data

Figure 9.19(a) shows typical hard disk format

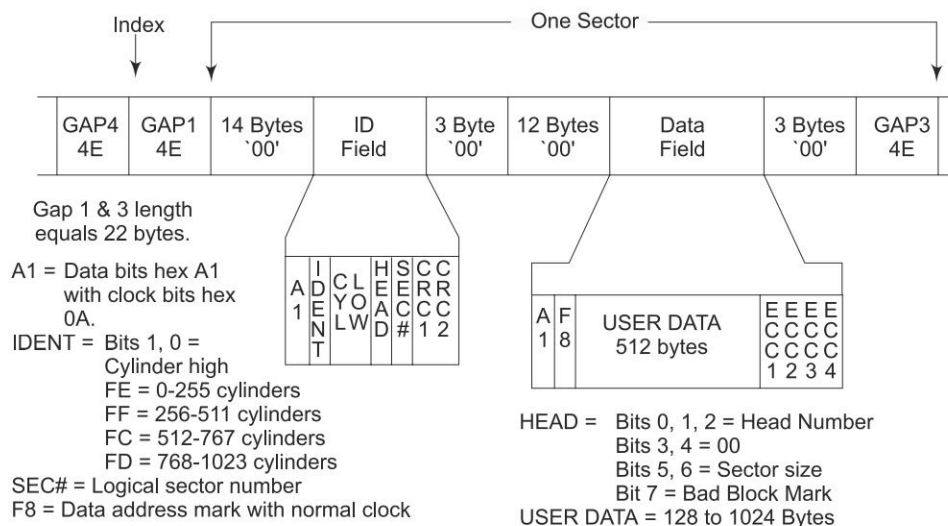


Fig. 9.19(a) Hard disk format

9.6.8.1 End to end Data Protection (EDP)

Advanced reliability is ensured by modern hard disk drives jointly with application software and the disk controller. To provide end to end data protection, three fields are added to each block of user data as shown in Fig.9.19(b). The EDP feature enables detection of errors in the entire path from the computer system to the disk media. The Guard is a two

byte field. It protects the data block content with CRCC. The application tag is a two byte field. This is an additional check information to each data block. The RAID systems and certain other applications make use of this feature to the application. The reference tag contains the block address information. It provides numbering of blocks so that verification of access to correct block is possible. All the three checks are created using a standard algorithm. Hence, a hard disk drive can check the received data (from the system) even during write operation and report errors in addition to error detection (and reporting) during read operation.

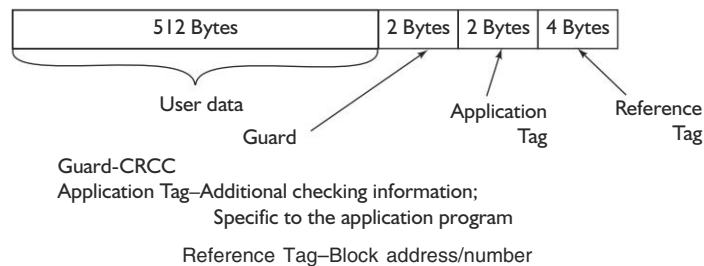


Fig. 9.19(b) End to end data protection

9.6.9 Case study 1: ST-506 Hard Disk Controller in IBM PC

The organization of the Hard Disk Controller (HDC) in IBM PC has close similarities to the FDC organization. The method of communication between the HDC hardware and the software (I/O driver) is same as the one followed for the FDC.

9.6.9.1 HDD Interface

The HDD interface used in the initial IBM PC is known as *ST-506* interface, initially adopted by *Seagate Technology*. To link the HDD to the HDC, two cables are used as shown in Fig. 9.20. The control cable is a 34-wire flat cable and carries two types of signals:

1. Control signals from the HDC
2. Status signals from the HDD

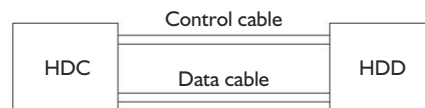


Fig. 9.20 Interfacing hard disk drive-ST-506

The Data cable is a 20-wire flat cable and it mainly carries WRITE DATA from the HDC and the READ DATA from the HDD. The interface connections for a two drive configuration is shown in Fig. 9.21. The control cable is common to both the HDDs and is termed as *daisy chain cable*. The data cable is separate for each drive and is known as *radial cable*.

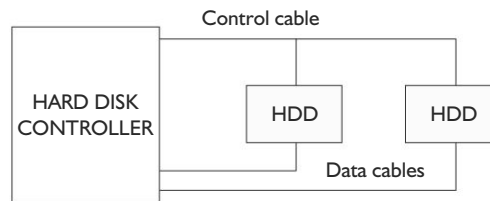


Fig. 9.21 Two HDDs configuration-ST-506

The drive number assignment is purely based on the DRIVE SELECT settings (jumpers or DIP switches) in the HDD. Any one of the drives is made as HDD 0 and the other as HDD 1.

9.6.9.2 Status Signals

Figure 9.22 illustrates the different status signals from the HDD. TRACK 0, INDEX, WRITE FAULT and DRIVE READY have meanings similar to the corresponding signals

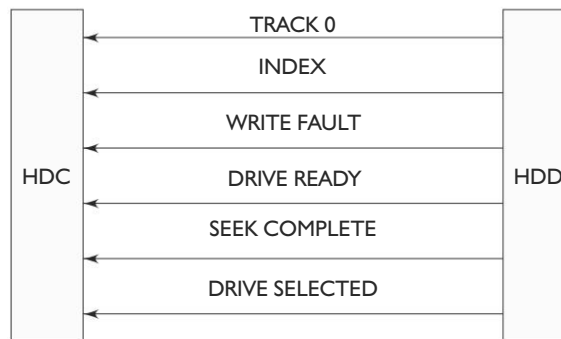


Fig. 9.22 HDD status signals

in the FDD signal cable. The SEEK COMPLETE signal indicates that a SEEK operation is completed. The DRIVE SELECTED is an acknowledgement signal from the HDD indicating that it is logically connected with the HDC. It is in response to the DRIVE SELECT control signal from the HDC.

9.6.9.3 Control Signals

Figure 9.23 illustrates the different control signals from the HDC. The signals DRIVE SELECT, STEP and DIRECTION have meanings similar to the corresponding signals in the FDD. The WRITE GATE is equivalent to WRITE ENABLE signal in the FDD interface.

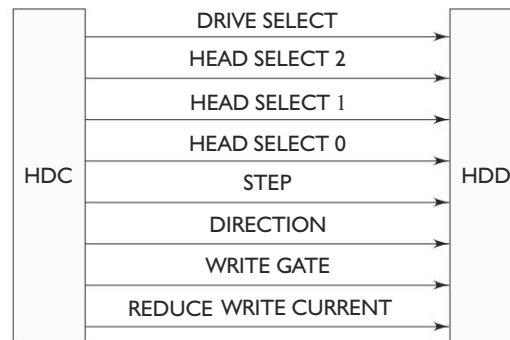


Fig. 9.23 HDD control signals

The REDUCE WRITE CURRENT signal is issued by the HDC when Read/Write heads in the HDD are positioned on the inner cylinders. Since the adjacent bits are more closer in the inner cylinders, interference between adjacent bits is more likely. As a precaution, the HDC instructs the HDD to reduce the “amount” (magnitude) of current through the read/write head so that the magnetic regions of adjacent bits do not interfere with each other. The three HEAD SELECT signals give the head number (in binary) indicating the head which should be involved in the current read or write operation.

9.6.9.4 Overview of HDC

The HDC performs data transfer with memory in DMA mode. The main features of HDC are as follows:

1. The HDC provides a data buffer with a capacity of one sector. This buffer isolates the main memory and the HDD as shown in Figs. 9.24 and 9.25. Hence, data transfer from the HDD to the HDC can be done at a faster rate.
2. The HDC has internal self-diagnostic capabilities. During power-on reset, the HDC performs self-test of its internal hardware.

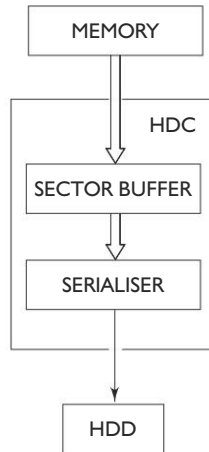


Fig. 9.24 Use of sector buffer during write operation

3. The HDC follows Error Checking and Correcting code (ECC) for the data field of each sector. While writing data, the HDC generates 4 bytes of ECC which are also stored in the data field. While reading data, the HDC regenerates ECC and matches with the ECC read. On mismatch, the HDC corrects the error in the data field.

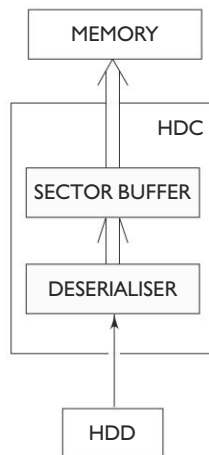


Fig. 9.25 Use of sector buffer during read operation

4. The HDC has an “automatic retry” feature. On encountering an error, while executing a command, the HDC automatically retries the command.
5. The HDC provides implied Seek feature. For a read/write command, it also performs seek operation if the read/write heads are not on the desired track.
6. The HDC supports two special commands: WRITE LONG and READ LONG. Using these, a diagnostic program can verify the proper functioning of ECC generation/checking circuits.
7. The HDC supports diagnostic commands for testing sector buffer, HDD and HDC hardware.
8. The HDC supports Read Verify Command for verifying the data on the HDD without transferring the data to the memory.
9. The HDC supports different types of HDDs. The INITIALISE DRIVE PARAMETERS command is used to inform the HDC of the drive parameters.

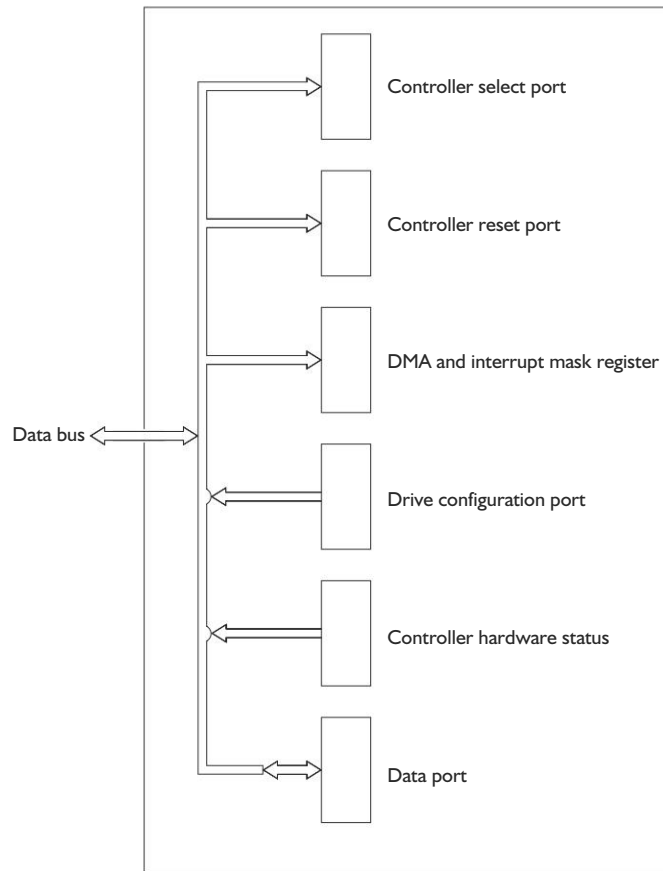
Communication between HDC and I/O Driver

As shown in Fig. 9.26, the HDC appears to be a set of I/O ports to the I/O driver: three output ports, two input ports and one input/output port. The “controller select” port is used by the I/O driver, to select the HDC before issuing a command to it. Just an OUT instruction, on this port, is sufficient to select the HDC. In response, the HDC sets the BUSY bit in the “controller hardware status” and is ready to receive the command. The “controller reset” port is used to reset the HDC. Like the “select port”, no data is transferred while addressing this port, and a mere OUT instruction on this port resets the HDC. The DMA and interrupt mask register port is used to suppress either the DMA request from the HDC or interrupt request from the HDC, or both.

Data Port

Data port is used for passing information between the I/O driver and the HDC as follows:

1. Reading command completion status from the HDC
2. Reading data from the HDC
3. Issuing data to the HDC
4. Issuing command and relevant command parameters to the HDC.

**Fig. 9.26** I/O Ports in HDC

HDC Status Port

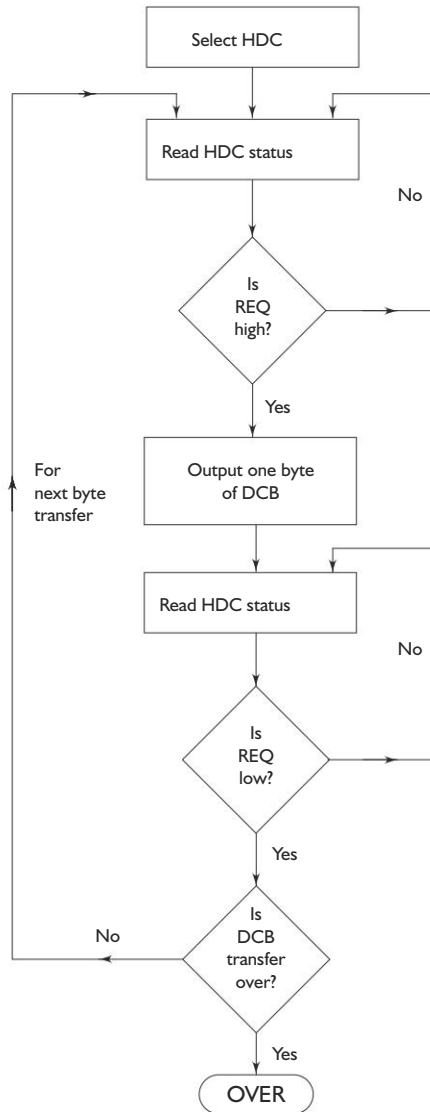
This port is used to read the overall status of HDC. This port is always accessible by the I/O driver irrespective of whether the HDC is free or busy.

Protocol between HDC and I/O Driver

The I/O driver must follow a specific protocol with the HDC. The command should be given in a specific format called the *command block* or *Device Control Block (DCB)* as a sequence of six bytes. On completion of a command, the HDC issues an interrupt to the CPU. Now, the I/O driver should read the command completion status from the HDC.

Issuing DCB to the HDC

The protocol followed by the I/O driver for issuing the DCB is illustrated in Fig. 9.27. As a first step, the I/O driver selects the HDC by an OUT instruction. The HDC responds by setting BUSY bit in the HDC status port. Now, the I/O driver issues the 6 bytes of DCB, one by one, by OUT instructions for the DATA port. For each byte, the I/O driver observes the protocol of REQ bit ON and OFF.

**Fig. 9.27** DCB issuing sequence

Sensing Command Completion

On completing a command, the HDC forms command completion status byte and raises interrupt. The I/O driver reads the command completion status through data port. In case of error, the I/O driver issues REQUEST STATUS command to find out the exact type of error. Now, the HDC presents four bytes of data known as *sense bytes* indicating the nature of the problem. The I/O driver should read these bytes before issuing another command.

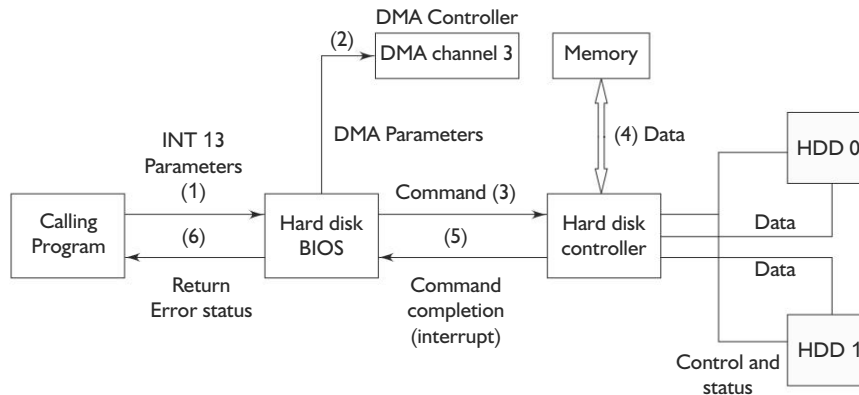
9.6.9.5 HDC Commands and I/O Driver

The HDC supports 19 commands. These commands and their functions are listed in Table 9.5.

TABLE 9.5 HDC commands and functions

Command	Function
Test drive ready	Tests whether the specified drive is ready and error-free
Recalibrate	Brings the read/write head in the specified drive to the Track 0
Request sense status	Presents four sense bytes relevant to previous command
Format drive	Formats the drive starting from the specified track
Ready verify	Reads data and checks for error
Format track	Formats the specified track
Format bad track	Formats the specified track and sets the 'bad block mark flag' in ID fields in all the sectors
Read	Reads data and transfers to the memory
Write	Transfers data from the memory and writes onto the specified sectors
Seek	Moves the read/write head in the HDD to the specified track
Initialise drive	Informs the drive type by supplying drive parameters to HDC
Read ECC burst error length	Provides one byte which indicates the length of the error encountered in the previous command
Read sector buffer data	Transfers data from the sector buffer to the main memory
Write sector buffer	Transfers data from the main memory to the sector buffer
RAM diagnostics	Tests the sector buffer.
Drive diagnostics	Tests the specified HDD without erasing the data
Controller Internal diagnostics	Tests the different hardware sections within the HDC
Read long	Reads data and ECC and transfers to main memory.
Write long	Transfers data and 4 bytes of ECC (per sector) from the memory and writes onto the HDD

Figure 9.28 illustrates the various hardware and software sequences involved in a hard disk operation. The encircled numbers indicate the order in which the events take place. It should be noticed that between the events 1 (call to the HDC BIOS) and 6 (Return from HDC BIOS), the chain of events 2, 3, 4 and 5 may have to be performed multiple times. The hard disk driver in the BIOS provides services related to the hard disk operation. These can be availed off by any software like OS, utility programs and user programs. The calling program and the BIOS follow a fixed protocol. The call to the hard disk driver are made using interrupt instruction (Int 13h). The hard disk driver (BIOS) supports different functions.



Note: The operations in the hard disk interface are not shown.

Fig. 9.28 Sequence of hard disk communication

While calling the hard disk driver-BIOS, the calling program passes relevant parameters through certain registers in the microprocessor. The hard disk driver (BIOS) takes these parameters from the registers and issues appropriate sequence of commands to the HDC. The exact parameters to be passed by the calling program to the hard disk BIOS varies with the function to be performed.

After completing the hard disk operation, the BIOS uses certain registers for conveying the result (status) of the operation.

9.6.9.6 HDC Organization

Figure 9.29 shows the functional organization of the HDC Hardware. The system interface includes the I/O ports and the DMA logic. The BIOS ROM contains the I/O drivers for

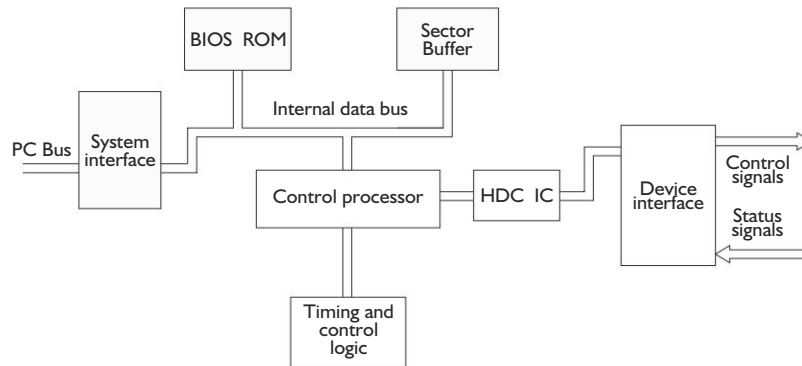


Fig. 9.29 Functional organization of HDC hardware

the HDC, self-test programs and certain utility programs. The sector buffer is used as a buffer memory between the main memory and the hard disk drive. During read or write operation, the sector buffer is used as a temporary storage for one sector data.

The control processor provides necessary logic so that the sector buffer is a dual port memory shared by both the system interface and the HDC IC. The control processor coordinates between the system interface, sector buffer and the HDC IC. In addition, its main function is decoding the command issued by the I/O driver and taking appropriate actions for the command. It also incorporates ECC generation and checking logic.

The HDC IC is a special disk controller IC such as Western Digital's WD1010 which is a powerful stand-alone disk controller. The device interface mainly consists of line drivers and line receivers to provide proper interfacing for the signals transmitted/received on the cables. The timing and control logic mainly generates clock signal and other timing signals.

In addition to sector buffer, there is a command buffer which is used for storing certain command parameters such as HDD drive characteristics and the type of disk formatting desired. Physically, the command buffer and sector buffer are part of a 2K static RAM in the HDC.

Command Block

For all commands, the format of the command block is same though certain fields are not relevant for some commands. The information provided by the command block are the following:

1. Opcode: Identifies the type and function of the command
2. Drive Number: Indicates the drive
3. Head Number: Indicates one of the heads
4. Cylinder Number: Indicates the cylinder
5. Sector Number: Indicates starting sector
6. Block Count (BLK): Indicates number of sectors
7. Interleave factor (INT): Indicates the interleave factor during formatting
8. General error should be retried or not
9. ECC error should be retried or not
10. Step code (SP): Specifies the rate at which STEP pulses should be issued to the HDD

Error Code

In order to find out whether a command is executed successfully without any error, the I/O driver reads the *command completion status* from the HDC, through data port. In case of error, to find out the exact nature of problem, the I/O driver issues REQUEST SENSE STATUS command to the HDC. In response, the HDC presents four sense bytes, of which the first byte is known as the error code which pinpoints the problem.

9.6.10 Traditional Hard disk interfaces in PC

The Hard Disk Controller (HDC) design has undergone significant changes in recent years. The following demands contributed to the development of new HDC models:

- Increased disk drive capacity
- Increased disk access time
- Increased data transfer rates
- Improved installation facilities

The initial HDC design supported disk drives with ST-506 interface. This was soon replaced by IDE controller which supports IDE drives. The IDE HDC is more of an interface adapter since, the 'real' controller has moved to the device. Though, the IDE provides increased drive capacity and higher data transfer rates, high performance systems such as servers use SCSI controller. The SCSI drives have both high capacity and faster transfer rates. The SCSI controller is an interface adapter like IDE controller, since, the SCSI drives have built-in controller.

The SCSI and IDE interfaces support non-hard disk peripherals also. The IDE Interface is used generally for CD-ROM drive and tape drive. The SCSI interface is used for a

variety of intelligent peripherals (upto 8) on single bus. Generally, magnetic tape and printers co-exist with the HDD on a single SCSI bus. The SCSI drives are costlier than IDE drives. The USB and FireWire are the modern interfaces that support hard disk as well as several other peripherals.

9.6.10.1 IDE

The Integrated Drive Electronics (IDE) interface is also known as AT Attachment (ATA) interface. The features of IDE are as follows:

1. The disk controller is part of the IDE hard disk. Hence, the interface adapter acts like a two way transceiver for the signals on the system bus.
2. A single 40-pin signal cable is used between the IDE adapter and the IDE hard disk.
3. Two IDE drives can be daisy chained to a single IDE adapter. One drive is set as the master and the other as slave. The controller in the master controls the slave also.
4. The maximum drive capacity and transfer rate for IDE are 528 MB and 10 Mbps.
5. *Drive Translation:* The drive geometry (drive parameters) supplied to the software set-up (configuration) need not match the physical data. The IDE drive supports mapping (translation) of logical parameters into physical parameters.
6. *Zoned Recording:* The IDE drive follows a variable track format. The tracks are divided into different zones. The tracks in the outer zones have more sectors per track whereas the inner zones have less sectors per track. This achieves more capacity for a given drive.

IDE Adapter

Figure 9.30 exhibits the block diagram for the IDE adapter. The IDE drive *has* 17 registers of two types: three control block registers *and* 14 command registers. The CS0FX is used to select the control block registers. The CS1FX is used to select the command registers. The exact operation (read/write) is indicated by DIOR or DIOW signal. The address bits DA2-DA0 identify individual register with-in a group.

9.6.10.2 EIDE

Enhanced IDE (EIDE) interface was developed to overcome the following limitations of IDE interface:

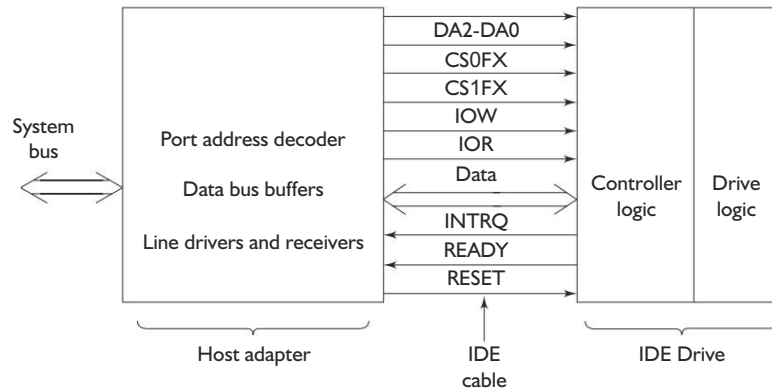


Fig. 9.30 Disk controller as part of disk drive

1. IDE supports mainly hard disk drives
2. IDE can handle a maximum of two drives
3. Maximum drive capacity for IDE drive is 528 MB
4. Data transfer rate is slow.

The EIDE interface can handle drives of capacity upto 8.4 GB with a maximum transfer rate of 16.6 MBps. It provides a primary EIDE channel and a secondary IDE channel. The primary EIDE channel can handle two EIDE drives in a master-slave configuration. The secondary IDE channel is meant for two non-hard disk IDE devices such as CD-ROM drive and tape drive.

9.6.10.3 SCSI

Small Computer System Interface (SCSI) is an universal parallel I/O interface to link multiple peripheral devices of different types on a single I/O bus. It is pronounced as 'Scuzzy'. Each device is allotted an address. The SCSI adapter in the PC performs an intelligent protocol sequence on the SCSI bus to which the connected SCSI devices respond. A 50-pin ribbon cable is used for connecting internal SCSI devices and thick shielded cable is used for connecting external SCSI devices.

The subsystems on the SCSI bus function in two different ways: *Initiator* and *Target*. An Initiator starts communicating with the target and the target responds to the command from the initiator. The SCSI adapter is an initiator and the peripheral devices are usually the targets. Upto eight subsystems can exist on a SCSI bus. The SCSI uses a handshaking protocol. A device connected to a SCSI is intelligent and hence expensive. Hence, SCSI

interface is used generally in servers and RISC systems. The principle of SCSI bus sequences is discussed in chapter 10.

9.6.10.4 USB and FireWire

The USB and FireWire are modern and intelligent interfaces that are applicable to hard disk also. The FireWire interface is more suited to hard disk drive than USB interfaces. The principle of USB interface is discussed in chapter 10.

9.6.11 Modern Hard Disk Interfaces

Some of the standard HDD interfaces are listed in Table 9.6.

TABLE 9.6 Typical hard disk interfaces

Sl no.	Interface Type Acronym	Description	Main feature/usage	Remarks
1	SASI	Shugart Associates System Interface	Predecessor to SCSI.	Obsolete today
2	SCSI	Small Computer System Interface	Bus oriented; handles concurrent operations.	In use
3	SAS	Serial Attached SCSI	Improvement of SCSI; uses serial interface.	In use
4	ST-506	Seagate Technology	Historical Seagate interface.	Obsolete
5	ST-412	Seagate Technology	Improvement over ST-506.	Obsolete
6	ESDI	Enhanced Small Disk Interface	Backwards compatible with ST-412/506, but faster and more integrated.	Obsolete
7	ATA	Advanced Technology Attachment	Successor to ST-412/506/ESDI ; integrates the disk controller inside the device.	Obsolete; Initially known as Integrated drive electronics (IDE)
8	SATA	Serial ATA	Modification of ATA, uses serial interface.	In use
9	FC	Fiber channel	A successor to parallel SCSI interface for enterprise systems. Has a serial protocol	Commonly used in storage area networks (SANs).

Example 9.1 A hard disk drive rotates at 7200 rpm. What is the rotational delay?

Average rotational delay is one half of the rotational speed.

Rotational speed = 7200 rpm

Time for one revolution = $1 \div 7200$ minutes = 8.33 msec

Average rotational delay is $(0.833 \div 2)$ msec = 4.16 msec

Example 9.2 Calculate the total capacity of a hard disk with following parameters:

- (a) Number of recording surfaces, h , = 20
- (b) Number of tracks per surface, t , = 15000
- (c) Number of sectors per track, s , = 400
- (d) Number of bytes per sector, B , = 512

Total capacity of hard disk = $h * t * s * B = 20 * 15,000 * 400 * 512 = 6144000000$ bytes
= 60 GB

Example 9.3 Calculate average disk access time for a hard disk drive with following parameters:

- (a) sector size = 512 bytes
- (b) rotation speed = 7200 rpm
- (c) average seek time = 12 msec
- (d) sustained transfer rate = 145 MB/sec
- (e) controller overhead = 1 msec

Average disk access time = Average seek time + Average rotational delay + transfer time
+ Controller overhead

Average rotational delay = 4.16 msec (as per Example 9.1)

Transfer time = $512 \text{ Bytes} \div 145 \text{ MB/sec} = 0.5 \text{ KB} \div 145 \text{ MB/sec} = 0.345 \text{ msec}$

Average access time = $12 + 4.16 + 0.345 + 1 \text{ msec} = 17.505 \text{ msec}$

9.7 RAID/DISK ARRAYS

RAID is an acronym to denote ‘Redundant Array of Inexpensive Disks’. It is a technique to obtain high levels of storage reliability using multiple low-cost (and less reliable) disk-drives, organised as an array. Presently, it has become common to use the term RAID to mean ‘redundant array of independent disks’. The term RAID is also now used as a general term for different storage techniques that split and replicate data among multiple hard disk drives. These techniques are identified by the word RAID followed by a number, as in RAID 1. RAID’s design objectives are increasing data reliability and/or increasing performance.

Though, the RAID array distributes data across multiple disks, the operating system recognises the array as a single disk drive since, the RAID controller manages the organisation.

In a RAID array, reliability is provided by either mirroring the same data in multiple drives, or writing extra information (such as parity data) or combining both techniques. As a result, failure of one (or more) disks in the array does not cause loss of data. A defective disk drive is replaced by a new one, and the lost data is rebuilt using the remaining data and the parity data. Many RAID systems support hot swapping: a drive is replaced live without powering-off the system

RAID Principles

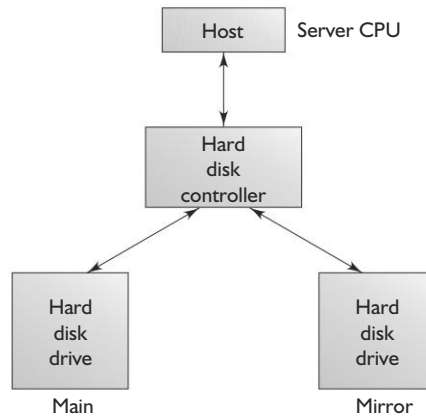
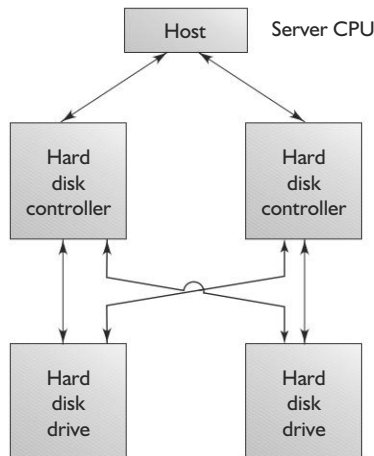
RAID combines two or more physical hard disk drives and presents as a single logical drive by using either special hardware or software. There are certain basic concepts used in RAID to provide data reliability/fault tolerance: disk mirroring (storing data in more than one disk drive, as shown in Fig. 9.31), disk duplexing (access to a hard disk drive via two disk controllers as shown in Fig. 9.32), data striping (writing data alternatively across multiple disks) and error correction (storing redundant data to enable error detection and correction). Different RAID levels use one or more of these concepts. Table 9.7 gives a brief summary of the popular RAID levels and fig. 9.33 presents a pictorial view of these levels.

TABLE 9.7 Standard RAID levels

RAID Level	Principle / Remarks	Merits	Demerits
0	"Striping". The data (file) is split into multiple parts. The number of parts equals the number of disk drives. The different parts are written on (Fig. 9.33(a)) respective disk drives simultaneously on the same sector number.	High performance at low cost	No redundancy; Even if one drive fails, it disables the entire array. No error checking; any error is unrecoverable.
1	"Mirroring". Each disk drive has a mirror disk drive; every byte is written on both drives (Fig. 9.33(b)) .	For every record, there are two copies. Very simple technique (at very high cost); good redundancy. Provides fault tolerance. The array is operational even if only one drive is working.	Expensive; maximum wastage of space

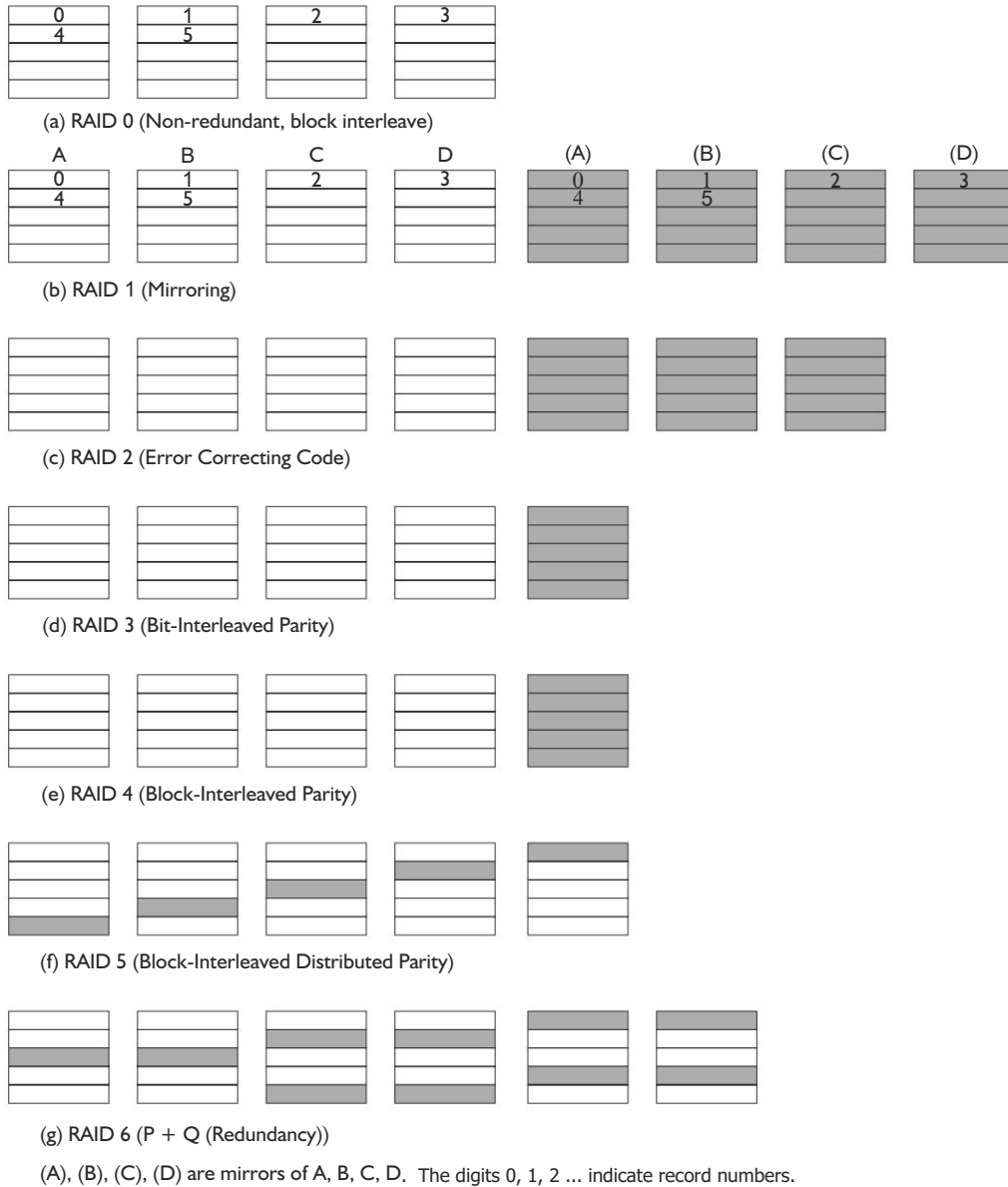
(Contd.)

2	Data striping with bit interleave and Hamming code. Disks striped in very small stripes. Each bit in a byte is written in a different disk drive. Separate drives for hamming codes. Hamming code stored on multiple parity disks. (Fig. 9.33(c))	1. High performance due to parallel write/read 2. Hamming codes provide error correction 3. No duplication of data	Expensive; not popular
3	Data striping with bit interleave and dedicated parity. Each bit in a byte written in a different disk drive; Separate drive for parity bit . (Fig. 9.33(d))	If one drive fails, the performance doesn't change. High performance; No duplication of data; Good reliability: when a failure is detected, recovery of lost data is done	The drives must have synchronized rotation.
4	Data striping with block interleave and dedicated parity. Each block of data written in a different disk drive; Separate drive for parity bit. Files can be distributed between multiple disks. (Fig. 9.33(e))	1. Each disk operates independently; multiple simultaneous operations. 2. Good reliability: error detection is through dedicated parity in a separate drive.	Expensive; a single disk drive for parity is a bottleneck.
5	Data Striping with block interleave and distributed parity check. Distributes parity evenly across all disk drives. The controller has necessary intelligence. Drive failure requires replacement, but the array is not affected by a single drive failure. (Fig. 9.33(f))	Same as above	Expensive
6	P + Q redundancy; uses Reed-Solomon codes. Data striping with dual distributed parity. Two different error codes are generated and stored (distributed) in two drives with block interleave data striping similar to RAID 5. (Fig. 9.33(g))	High redundancy; Up to two drives failure does not affect the array; it continues to operate.	Expensive and additional delays.

**Fig. 9.31** Disk mirroring**Fig. 9.32** Disk duplexing

Nested (hybrid) RAID

In hybrid RAID, the RAID levels are nested. For example, RAID 10 (or RAID 1+0) consists of multiple level 1 arrays of disk drives, each of which is one of the “drives” of a level 0 array striped over the level 1 arrays.


Fig. 9.33 RAID levels

Software RAID

Software RAID implementation is achieved by the operating systems at low cost. A software layer above the disk device driver acts as an abstraction layer between the logical drives (RAIDs) and physical drives. Most common levels are RAID 0 (striping across multiple drives) and RAID 1 (mirroring two drives), RAID 1+0, RAID 0+1, and RAID 5 (data striping with parity). Software RAID gives poor performance compared to hardware RAID. Certain software RAIDs use different partitions of a single drive instead of having multiple physical drives.

Hardware RAID

The hardware RAID controller shares processor resources. It offers better error handling than the software RAID. The RAID controller and the disk drives are usually in a stand-alone disk enclosure (instead of placing inside the computer cabinet). The enclosure may be directly attached to the computer, or connected via SAN. The controller hardware handles the management of the disk drives, and performs error detection/correction relevant to the RAID level.

The hardware RAID offers high performance, and can support multiple operating systems. It also supports hot swapping, allowing replacement of failed drives while the system is running.

Firmware/driver-based RAID

Certain cheap “RAID controllers” do not contain the standard RAID controller, but use a standard hard disk controller with special drivers.

Network-attached Storage

Network-attached storage (NAS) is a dedicated equipment containing disk drives and is accessible over a computer network.

Hot Spares

RAIDs commonly support the use of hot spare drive, an additional drive installed in the array which remains idle usually. When an active drive fails, the system automatically substitutes the failed drive with the spare drive and restores the array.

Blade Server

The blade server is a combination of computer servers with a modular design optimized to minimize the use of physical space. The basic concept is 'All in one': integrate servers, storage, networking and I/O into a single chassis.

The blade server has special techniques to save space, minimize power consumption and other considerations. A blade enclosure houses multiple blade servers and provide common services such as power, cooling, networking, various interconnects and management. The blades and the blade enclosure together form the blade system.

In the blade server concept, most of the functions other than basic I/O and processing are relieved from the blade computer. These are either provided by the blade enclosure (e.g. DC power supply), virtualized (e.g. iSCSI storage, remote console over IP) or eliminated entirely (e.g. serial ports). The blade computers become simpler, smaller and cheaper.

The ability to boot the blade from a storage area network (SAN) enables a disk-free blade. The space thus saved, can be used for increased memory or additional CPUs.

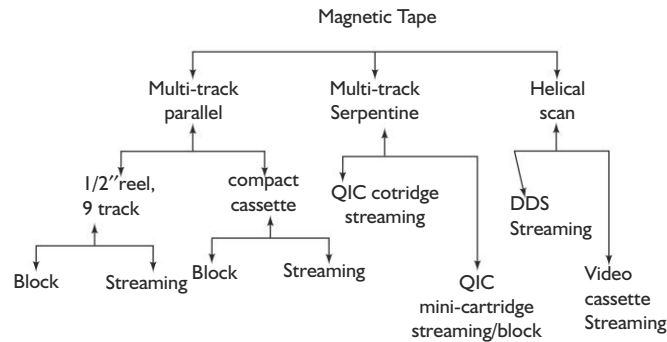


9.8 Magnetic Tape Data Storage

Magnetic tape has been used for data storage for over 50 years, mostly with large computer systems. In the early days, disk storage was small and expensive; hence, tape was used for data processing and storage. Presently, the application of tape is for archive and backup, since tape is less expensive than disk for storing large quantity of data.

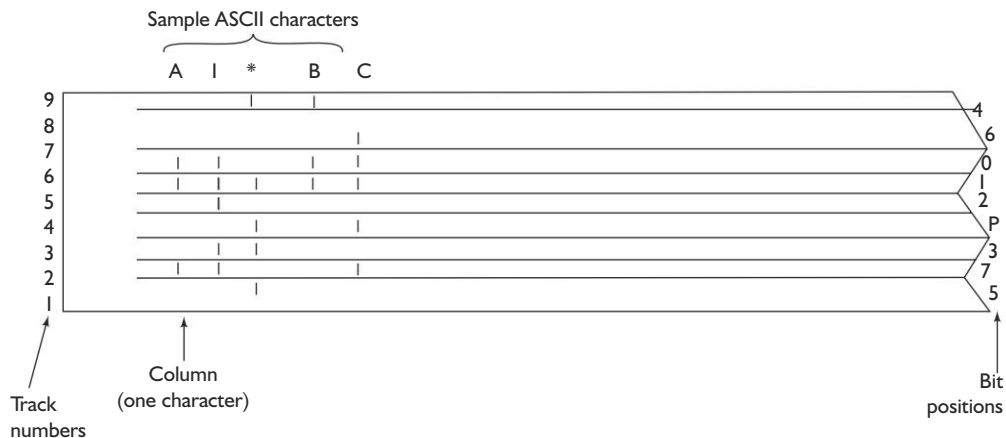
Initially, magnetic tape was wound on large (10.5 in) reels. Modern magnetic tape is packaged in cartridge and cassettes. The cassette's enclosure holds two reels with a single span of magnetic tape. The cartridge has a single reel of tape in a plastic enclosure; the tape drive has the takeup reel. A tape drive has motors to wind the tape from one reel to the other. A special type of cartridge has tape wound on a special reel in which tape can be removed from the center of the reel. There is no take-up reel inside the tape drive. Modern cartridge formats include DAT/DDS, DLT and LTO with capacities varying from tens to hundreds of gigabytes. Recently, 1 TB capacity tape cartridge has been introduced.

Fig. 9.34 shows different types of tape drives. As seen from the figure, some drives operate in block modes, whereas others operate in streaming mode. Some drives have fixed heads whereas others have rotating heads.

**Fig. 9.34** Classification of magnetic tape drives

9.8.1 Recording Method

There are three popular recording methods: linear, linear serpentine and scanning. In the linear method, data is stored in parallel tracks along the length of the tape. Fig. 9.35 shows a 9-track tape and recording format. There is one head for each track. All heads simultaneously write in respective tracks. It is a simple recording method, but provides low data density. In the linear serpentine recording, there are more tracks than number of heads. On some tracks, writing is done in forward direction and in remaining tracks, in reverse direction. While writing in one direction, after making a pass over the whole length of the tape, all heads shift slightly and make another pass in the reverse direction, writing another set of tracks. This is repeated on remaining tracks. The data storage capacity is higher compared to simple linear method.



Bit positions and track numbers are not identical
P-parity bit on track 4. odd parity is followed

Fig. 9.35 9-track tape recording pattern

Scanning method writes short tracks across the width of the tape (not lengthwise). The heads are present on a drum or disk which rotates fast and the slow moving tape passes it. In transverse scan method, the heads are mounted on the outer edge of a spinning disk that is positioned perpendicular to the path of the tape. In arcuate scan method, the heads are on the face of the spinning disk which is laid flat against the tape. The path of the tape heads makes an arc. Helical scan method writes in diagonal manner. This recording method is used by videotape systems and several data tape formats.

Sequential access /access time : Since, the tape is a sequential access medium, files are identified by number (not by filename). New data is generally included by adding a file to the end of the existing recording and not by overwriting a particular file (or part of file). Tape has a long latency since, the drive must wind a long length (most of the time) of the tape to move from one block to another. This is compensated in one of the two ways:

1. using indexing: a lookup table maintains the physical tape locations for data blocks
2. using tapemarks for the blocks: the tape mark is sensed while winding the tape fast.

Data compression

Most tape drives use data compression. A ratio of 2:1 is typical. Software compression gives better ratios, but slow. Some enterprise tape drives encrypt data after compression.

Though the seek time is high, tape drives stream data quickly. Modern LTO drives provide data transfer rates of up to 80 MB/s, which is equivalent to 10,000 rpm hard disks. Tape autoloaders and tape libraries assist in loading, unloading and storing multiple tapes to further increase archive capacity. Tape drives are commonly interfaced to a computer via SCSI, Fibre Channel, SATA, USB, FireWire, etc.

9.8.2 Start-Stop Blocked Mode

In this mode of operation, data is recorded in fixed blocks as shown in Fig. 9.36. Between two adjacent blocks, there is an Inter-Record Gap (IRG) of up to 0.5 inch. The tape controller uses the IRG to locate the beginning of each block and can update or over-write the data block wise. Tape movement has sequences of forward- stop, and reverse-stop operations in quick succession. Also, the tape can search for a block by skipping in the forward or reverse direction. Each block is written with the tape running continuously. The shoe-shining effect occurs during writing (or reading) data, if the transfer rate of the data falls below the minimum threshold at which the heads are designed to transfer data. When this occurs, the

drive must stop the tape, rewind back a little, restart to a proper speed and continue writing from the same position. When shoe-shining occurs, it affects the data rate. In addition, shoe-shining causes stress on the drive mechanism and the tape medium, increasing hardware failure rate.

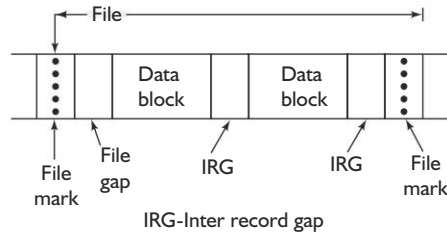


Fig. 9.36 Start-stop blocked mode data organization

Modern tape drive designs have internal data buffer. The tape is stopped only when the buffer has no data to be written (underrun), or when it is full during reading. The host computer chooses appropriate block size. There is a trade-off between the block size, the size of the data buffer, the extent of tape wasted on inter-block gaps, and read/write throughput. Modern drives have multiple speed levels. They dynamically match the tape speed level to computer's data rate. Typical speed levels are 50%, 75% and 100% of full speed.

The blocked mode is used in 1/2-inch fixed head multi-track drives and in few mini-cartridge units. In the mini-cartridge units, additional software formats the tape into "tracks" and "sectors" to emulate a floppy disk drive of 20/40 Mbytes.

IBM format

Initially, IBM used ferrous-oxide coated tape—similar to audio recording. The tape was 0.5" wide wound on reels of 10.5 inches in diameter. Different lengths were available with 1200', 2400' or 3600'. In IBM System 360 mainframe, 9 track tape was used. Recording densities of 800, 1600, and 6250 cpi were used. This amounts to about 5 MB to 140 MB for 2400 ft tape. *End of file* was designated by a tape mark and *end of tape* by two tape marks. Early IBM tape drives used vacuum columns to hold long U-shaped loops of tape. The two tape reels fed tape into or pulled tape out of the vacuum columns. Figure 9.37 shows a typical vacuum column magnetic tape drive.

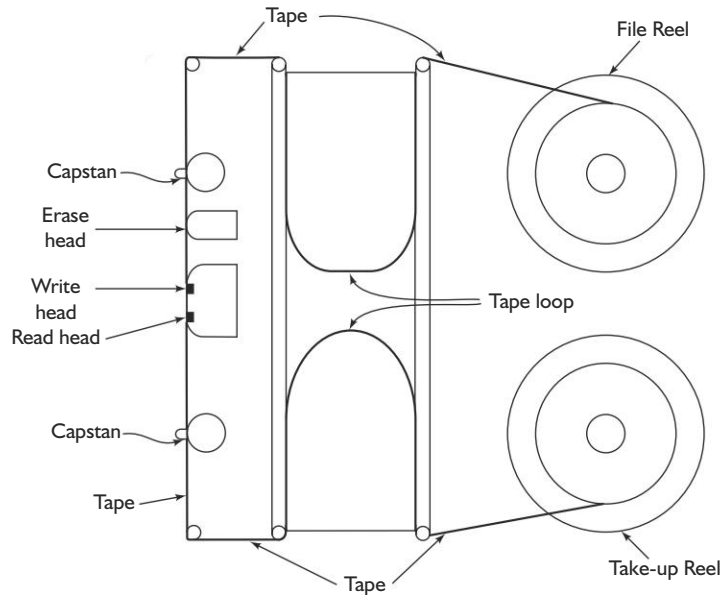


Fig. 9.37 Half-inch vacuum column magnetic tape drive

9.8.3 Streaming Mode

For archiving and backup, a file or set of files are stored on the tape. If these files are modified, the updated versions are stored by re-writing the complete files. For such applications, streaming mode tape drives are used. Streaming tape drive writes complete files as a continuous “stream” of data. It is not possible to locate and modify a particular block. Tape motion is in the forward direction for operations such as read, write or skip forward to file mark or end of file. Reverse motion is used only to rewind the tape back. There are two types of streaming mode tape drives: fixed head and rotary head drives.

9.8.3.1 Fixed Head Streaming Mode Tape Drives

In early digital tape drives, normal audio recorders were used for storing by using the data to modulate tone signals in the audio frequency range. FSK, PSK and other modulation schemes have been used. However, digital tape drives optimize data storage density and transfer by several techniques. In order to achieve high storage efficiency, direct digital recording is used. The tape is moved fast across the head to get high data transfer rate.

Digital tape drives run the tape at over 45 ips. Multiple tracks are recorded along the length of the tape simultaneously with multiple recording heads.

The QIC Standard

The QIC (Quarter-Inch Cartridge) industry standard specifies a fixed head, streaming tape. It has two sets of read/write heads, one for each direction. Parallel tracks are obtained by physically moving the head perpendicularly across the width of the tape to access a new track. Each track is written sequentially resulting in a serpentine pattern.

QIC defines a number of standards: QIC-24, QIC-150, QIC-525, QIC-1350 etc. differing in capacity. QIC-24 standard is briefly covered below.

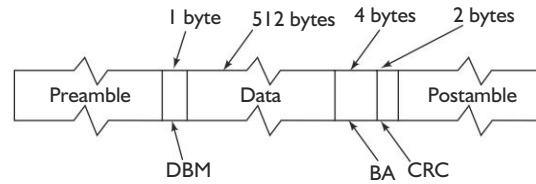
QIC-24 Cartridge Tape Drive

The Write Operation sequence is listed below and read operation is similar.

1. When the tape is loaded, rewinding occurs. Tape is brought to the Beginning of Tape (BOT), and head is positioned for Track 0.
2. For WRITE command, tape is moved to the Load Point (LP). This is the start of the recording zone; drive begins writing on Track 0. Data is written bit serial on one track. In addition, during the writing on Track 0, the erase head is enabled. Full track erase is done by applying AC signal to the erase head ahead of the write head.
3. As soon as the Early Warning (EW) hole is detected, the drive stops accepting data from host computer. Recording stops after the remaining data in buffer is written.
4. Tape movement continues in the forward direction until End of Tape (EOT) is reached. Erase head is disabled. The head for Track 1 in reverse direction is selected.
5. Tape is now moved in the reverse direction and recording begins when EW hole detected. Recording continues until the LP hole is detected.
6. On reaching BOT, the head is moved (from Track 0 to Track 2), and positioned at the next track. Head is moved across the width of tape to various tracks by a stepper motor.
7. Tape movement is changed back to the forward direction and Track 2 is written.
8. This procedure is continued, changing directions and shifting the head after every pair (F/R) of tracks creating the serpentine recording pattern.

Data Block Format

Figure 9.38 shows the format of the data block used in the QIC-24 tape drives.



DBM-Data Block Marker
 BA-Block Address (Track number and block number)
 CRC-Cyclic Redundancy Check

Fig. 9.38 QIC data block format

- Preamble: This is a sequence of bytes to synchronize the PLL to the data frequency.
- Data Block Marker: GCR code '11111 00111'
- Data Block: The 512 data bytes encoded as GCR 4/5.
- Block Address: The block address consists of a 4-Byte block address with track number, a control nibble and block number starting with 1 on each track.
- Cyclic Redundancy Check Character: A 2-Byte CRC character is calculated over the 512 bytes of data and 4 bytes of block address using the polynomial: $x^{16} + x^{12} + x^5 + 1$
- Postamble: A postamble of 5 – 20 flux transitions at maximum density acts as a guard band. Elongated postamble of 3500 – 7000 flux transitions is used if underrun occurs.

QIC-02 Commands

Table 9.8 gives the QIC-02 command set. Figure 9.39 gives the QIC interface signals.

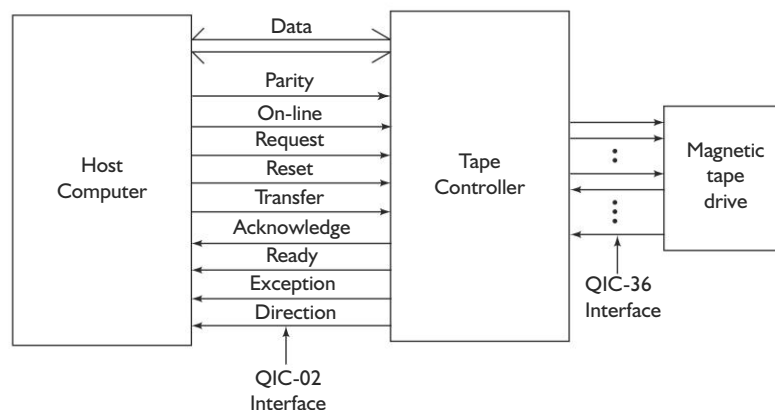


Fig. 9.39 QIC Interface signals

TABLE 9.8 QIC-02 standard command set

Sl no.	Command Code (Hexa decimal)	Command
1	01	Select drive, Soft lock OFF
2	11	Select drive, Soft lock ON
3	21	Position to beginning of tape (BOT)
4	22	Erase entire tape
5	24	Retension cartridge
6	40	Write
7	41	Write without under run
8	60	Write filemark (FM)
9	80	Read
10	81	Space forward
11	89	Space reverse
12	A0	Read filemark
13	A3	Seek end of data
14	Bn	Read n filemarks
15	C0	Read status
16	C2	Run self test 1
17	CA	Run self test 2

QIC Tape Drive Electronics

Figure 9.40 shows the block diagram of a QIC tape drive with a SCSI interface. A microprocessor/microcontroller and the custom VLSI, provides the control of the drive. A control program resides in EPROM. The DRAM is used for buffering. The standard SCSI controller operates under the control of the microprocessor, negotiating handshake protocols, commands and data transfer with the host computer. The VLSI takes care of a number of functions as follows:

1. DMA Controller: data transfer with the host through the SCSI controller.
2. Interrupt Controller: interrupt logic
3. Memory Access Controller: multiplexing of address, DRAM refresh and parity check.
4. Read/Write Controller: performs read-after-write to verify correctness of writing.
5. Clock Generator: clocks to microprocessor, SCSI interface and read/write phase clocks.
6. Data Separator: During read operation, helps separating data from clock using PLL.

For write operation, the erase circuit is active during the write of track 0, when erase is performed on the whole tape. The write circuit controls the write current to cause the flux transitions on the tape.

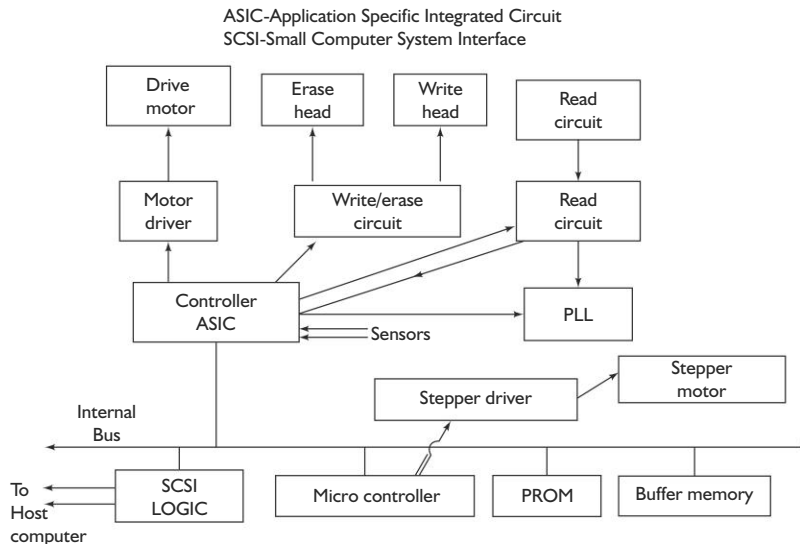


Fig. 9.40 QIC tape drive block diagram

9.8.3.2 Rotary Head Streaming Mode Tape Drives

The principle of rotary head recorders is the same as the video cassette recorders. One type of this group is based on the video cassette recorders and the other is rotary head digital audio tape (RDAT). In Rotary head recorders, instead of moving the tape, the head is rotating at a high speed across the tape. The tape also moves slowly at a rate sufficient to feed the head as it rotates. The tracks are allowed to overlap, and cross-track interference is reduced by offsetting the azimuths of the multiple heads.

Typical specifications of a DAT product are given below:

Specifications of a DDS DAT drive — Product: Archive Python 4330XT

Capacity: 1.3 GBytes with 60m tape.

Sustained transfer rate: 183 Kbytes/sec, sustained.

Average access time: 20 sec. seek time.

Form factor: 3 1/2"

Recording format: ANSI DDS

Interface: SCSI-1 and SCSI-2

Media: 4 mm. DAT Cartridge, 60/90m length.

Packing density: 1869 tracks/in.

Arial density: 114 Mbits/sq. in.

Uncorrectable error rate: Using ECC, 1 in 1015 bits.

Drum rotation speed: 2000 RPM

Tape speed: 0.32 in/sec.

Search/rewind speed: 200 X normal speed

Head-to-tape speed: 123 in/sec, helical scan (RDAT)

9.9 Optical Disk Storage

Optical disk storage is a recent technology compared to magnetic storage. Initially, it was used for audio recording and subsequently, it has become popular for data storage and video recording. Whereas magnetic storage devices use changes in magnetic polarity, for reading data, the optical storage devices use, variations in light reflection. An optical disc drive uses laser light for reading or writing data. The optical drive is rapidly replacing floppy disk drives and magnetic tape drives for archival use because of the low cost of optical media. Compact discs, DVDs and Blu-ray discs are common types of optical media.

The optical media has two major advantages over magnetic storage disk: high storage density and high reliability. The optical media can store nearly ten times the data stored in magnetic media of the same size. The data stored in optical media is more permanent compared to magnetic storage. The optical media is not affected by temperature variations, magnetic decay or electromagnetic interference. The major drawback of optical media is its poor access speed. Table 9.9 compares the magnetic media and the optical media.

TABLE 9.9 Comparison of magnetic and optical storage media

Sl no.	Parameter	Floppy disk	Hard disk	CD-ROM	DVD
1	Capacity for same space	1.44 MB	2 TB	650 MB	4.7 GB
2	Transfer rate	36 KB/s	1000 KB/s	100 KB/s	21.13 MB/s

The optical storage medium is a disc of highly reflective material encased in a plastic layer. The following are popular optical media types.

1. Compact Disc-Read only Memory (CD-ROM): Contains permanent programs/data stored by manufacturers. The users can only read from it but can not store data.
2. Compact Disc- Recordable (CD-R): Blank CD on which users can write once only.
3. Compact Disc- Read Writable (CD-RW): Users can write any number of times
4. Video CD (VCD)
5. Digital Versatile Disc (DVD)

9.9.1 CD-ROM

The Compact Disc is formed of a polycarbonate substrate, a reflective metalized layer, and a protective lacquer coating. Fig. 9.41 shows the cross section of a CD-ROM. During manufacturing, an injection-molded piece of clear polycarbonate plastic is impressed with microscopic ‘bumps’ arranged as a spiral track of data. Then, a thin reflective aluminum layer is sputtered, covering the bumps. Next, a thin acrylic layer is sprayed over the aluminum to protect it. The label is then printed onto the acrylic. The data is stored as a series of tiny indentations (pits), encoded in the spiral track mold. The areas between pits form the lands. NRZI encoding is used instead of actual binary data : a change from pit to land or land to pit indicates a one, while no change indicates a zero.

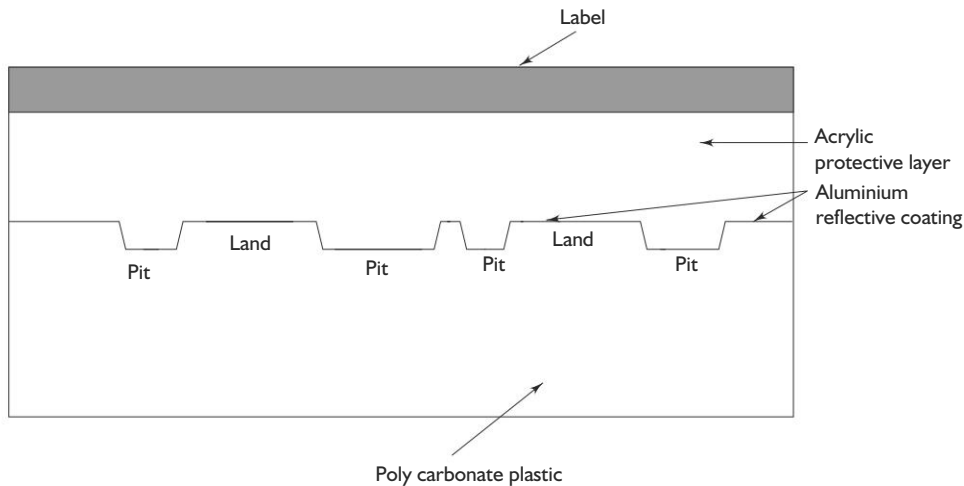
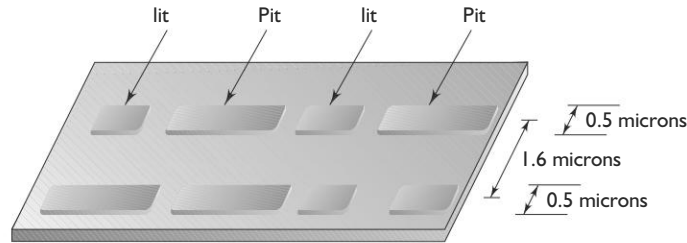


Fig. 9.41 Cross section of a CD media

Fig. 9.42 shows the appearance of the bumps looking through the polycarbonate layer. The bumps are each 0.5 microns wide, a minimum of 0.83 microns long and 125 nanometers high. The bumps appear as ‘pits’ on the aluminum side, though they are bumps on the side the laser reads from. The spiral track (Fig. 9.43) of data travels from the inside of the disc to the outside and is almost 3.5 miles (5 km) long. It is approximately 0.5 microns wide, with 1.6 microns gap to next track. On CD-ROM, the groove, made of pits, is pressed on a flat surface (land), during the manufacturing process. While reading, because of the depth of the pits, the reflected beam’s phase is shifted in relation to the incoming beam; this results in mutual destructive interference and reduces the reflected beam’s intensity. This is detected by photodiodes.

**Fig. 9.42** View of the bumps

Standard CDs are available in two sizes. The most common is 120 mm disc, with a 74- or 80-minute audio capacity and a 650 or 700 MB data capacity. 80 mm disc known as “Mini CDs” (21 minutes of music or 184 MB of data) is less popular. Table 9.10 lists different size CDs.

TABLE 9.10 CD sizes and capacities

Sl no.	Size	Audio Capacity	Data Capacity	Remarks
1	12 cm	74–99 min	650–870 MB	Standard size
2	8 cm	21–24 min	185–210 MB	Mini-CD size
3	85 × 54 mm – 86×64 mm	~6 min	10–65 MB	Business card size

Two main servomechanisms are used in a CD-ROM drive:

1. Maintaining a correct distance between lens and disc to ensure that the laser beam is focused on a small spot.
2. Moving the head along the disc’s radius to keep the beam on a spiral groove.

9.9.1.1 Read Mechanism

A CD is read by focusing a 780 nm wavelength (near infrared) semiconductor laser through the bottom of the polycarbonate layer. Due to the variations between pits and lands, light is either scattered by the disc surface or reflected back into a detector. The beam is reflected back into the detector from a landing and scattered by a pit. The change in height between pits and lands results in a difference in intensity in the light reflected. By measuring the intensity change with a photodiode, the data is read from the disc.

9.9.1.2 Tracking

Maintaining the laser beam centered on the data track is done by the tracking system. The tracking system continually moves the laser outward. As the laser moves outward from the

center of the disc, the bumps move past the laser faster on outer tracks. Hence, the spindle motor must reduce the speed of rotation of the CD as the laser moves outward. Due to regulation of the speed, the bumps travel past the laser at a constant speed. Hence, data is read at the same speed irrespective of the laser detector's position. The rotational mechanism in CD drive must achieve a constant throughput. Hence, the drive operates with a constant linear velocity (CLV). The disc angular velocity is not constant, and spindle motor varies speed between 200 RPM on the outer rim and 500 RPM on the inner rim.

9.9.1.3 CD-ROM Interfaces

The CD-ROM drive is interfaced to today's computers in many ways:

Via sound cards with a second ATA interface to support an optical drive

Via a SCSI interface

Via parallel port for optical disk drive; an option for laptops

Via a PCMCIA optical drive interface for laptops

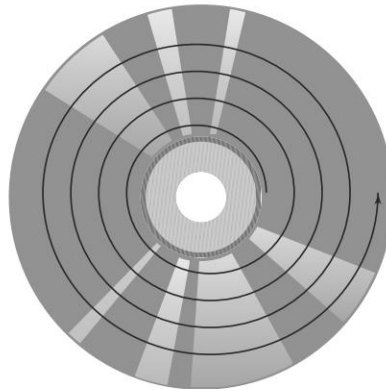


Fig. 9.43 CD-track organization

9.9.2 CD-Recordable (CD-R)

CD-Recordable media is used to record data using a recorder. The CD-R media can be written only once. The CD-R recording is permanent and its practical life varies from 20 to 100 years. During the manufacturing process, a “blank” data spiral is created in the CD-R by injection moulding. It has an organic dye as the data layer between the substrate and the metal reflective layer. The recorder burns data with a laser, by selectively heating parts of the organic dye layer. This changes the reflectivity of the dye and forms marks that can be

read (similar to the pits and lands on CD-ROM). The writing laser is more powerful than the reading laser.

Packet writing scheme is used by the recorder to write short packets at different times. Finally, the CD can be closed by writing a table-of-contents at the beginning of the disc. After closing the disc, further writing cannot be done. With appropriate software, the packet writing enables the CD to behave like a random write-access media as with flash memory and magnetic disks.

9.9.3 CD-Rewritable (CD-RW)

CD-RW is a re-recordable medium. It uses a metallic alloy instead of a dye. The write laser heats and alters the properties of the alloy thereby changing its reflectivity. The laser melts the crystalline metal alloy in the recording layer. Depending on the amount of power applied, the substance may melt back into crystalline form or remain in an amorphous form, creating spots of varying reflectivity. A low power reads the information. A very high power records the data bits. A medium power puts the alloy in crystalline state.

Fixed-length packet writing divides the disc into padded, fixed-size packets. The padding enables the recording on an individual packet without affecting its neighbours. This is equivalent to the block-writable access of the magnetic media. Such discs are not readable in most CD-ROM drives directly; use of additional I/O driver is needed.

9.9.4 DVD

Digital Versatile Disc (DVD) or Digital Video Disc, is an optical disc storage media for video and data storage. DVD has the same dimensions as compact disc (CDs), but stores more than six times.

There are many variations of DVD : DVD-ROM (read only memory) can be read but can not be written; DVD-R can record data only once; DVD-RW and DVD-RAM can record (and erase) data multiple times. DVD-Video and DVD-Audio discs refer to video and audio content respectively. Other types of DVDs are termed as DVD Data discs. The videodisc uses 30-cm disk for video recording. It uses an analog technique, using a laser to read a variable-width track similar to a phonograph record.

9.9.4.1 Dual-layer Recording

The dual-layer disc has two layers. The drive accesses the second layer by shining the laser through the first semitransparent layer. Up to 8.54 GB of data per DVD-R and DVD+R disc is possible in dual-layer recording compared with 4.7 GB in single-layer disc.

9.9.5 Future Optical Products

1. Blu-ray disc (BD) designed by Sony, Philips, and Panasonic; a dual layer offer 50 GB
2. HD-DVD
3. Holographic Versatile Disc (HVD) offer 500GB. It employs a technique known as collinear holography
4. The 5D DVD, being developed at the Swinburne University of Technology in Australia, uses a multilaser system on multiple layers. Disc capacities are estimated at up to 10 terabytes.

SUMMARY

Several types of secondary storage have been introduced in various types of computer systems from desktop PC to enterprise server. Magnetic storage devices and optical storage devices are two major types of secondary storage. Floppy disk and Hard disk drives are magnetic storage devices with rotating disks. Magnetic tape has been used for data storage in the early days, for data processing and storage. Presently, the application of tape is for archive and backup, since tape is less expensive than disk for storing large quantity of data. Initially, magnetic tape was wound on large (10.5 in) reels. Modern magnetic tape is packaged in cartridge and cassettes. The optical media has two major advantages over magnetic storage disk: high storage density and high reliability. The major drawback of optical media is its poor access speed. Presently, there are three versions of optical media: CD-ROM, CD-R and CD-RW. The DVD is an optical disc storage media for video and data storage. DVD has the same dimensions as compact disc (CDs), but stores more.

EXERCISES

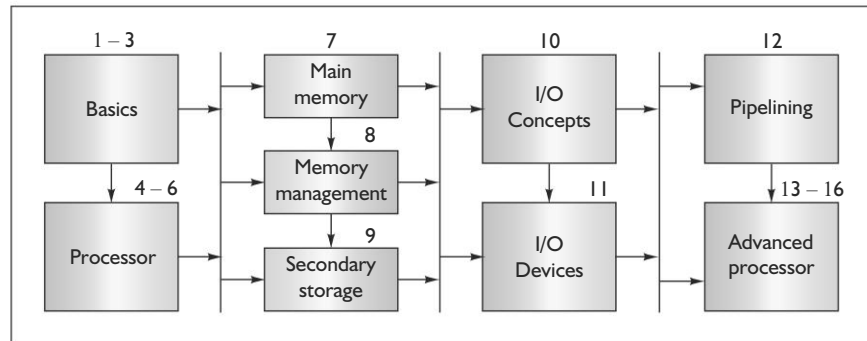
1. A hard disk drive rotates at 10,000 volutions per minute. What is the rotational delay?
2. Calculate the total capacity of a hard disk with following parameters:
 - (a) Number of recording surfaces = 5
 - (b) Number of tracks per surface = 697
 - (c) Number of sectors per track = 20
 - (d) Number of bytes per sector = 512
3. Calculate average disk access time for a hard disk drive with following parameters:
 - (f) sector size = 512 bytes

- (g) rotation speed = 3600 rpm
 - (h) average seek time = 45 msec
 - (i) sustained transfer rate = 5 MB/sec
 - (j) controller overhead = 2 msec
4. A computer program repeatedly does the following tasks: read a 4 KB block from the hard disk, process the data and generate a 4 KB block, and store the new block on the disk. The program is fully dedicated for the above tasks without any overlap between the I/O operations and processing. The processing task takes 10 million clock cycles and the processor clock rate is 800 MHz. The data is organised as a continuous stream on a track. Determine the overall performance of the system in terms of blocks processed per second. Assume the following parameters for the hard disk drive:
- (a) disk rotation speed = 10000 rpm
 - (b) transfer rate = 145 MB/sec
 - (c) controller overhead = 1 msec
 - (d) average seek time = 12 msec
5. A hard disk drive has a latency of 4.16 ms and a burst bandwidth of 10 MB/s. The CPU reads a 4 KB of data. Calculate the overall data transfer rate.

CHAPTER 10



I/O CONCEPTS AND TECHNIQUES



Chapter Flow Diagram

OBJECTIVES

- « Input/Output organization
- « Accessing I/O devices
- « Interrupts
- « Bus structures
- « Interface circuits
- « Standard I/O interfaces
- « I/O techniques
- « I/O ports
- « Programmed mode I/O
- « Interrupt mode I/O
- « DMA mode I/O
- « I/O processor
- « Synchronous and asynchronous transfer
- « Bus arbitration

KEYWORDS

Local Bus	Bus Arbitration
System Bus	Cycle Stealing
Mezzanine Bus	SCSI
I/O Bus	USB
I/O Mapped I/O	FireWire
Memory Mapped I/O	Wait State
Interrupt Nesting	RS-232C Interface
Device Controller	Centronics Interface
I/O Latency	IDE Interface
I/O Bandwidth	DMA
Daisy Chaining	PCI Bus
Handshaking	System Interface
Multiplexor Channel	Parallel Interface
Selector Channel	Serial Interface
Burst Mode	Data Channels

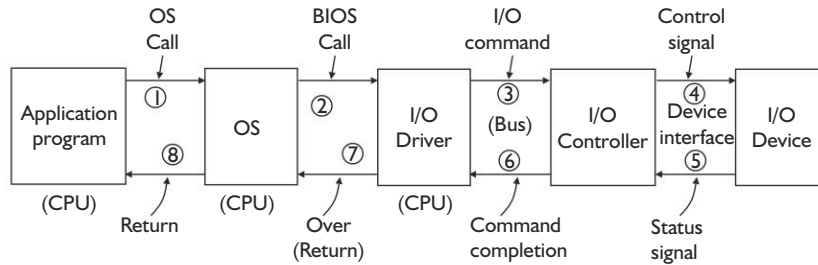
 10.1 Introduction

The processor communicates with the input and output units both for the user and system requirements. The performance of a computer system varies with the methods used for communication between the functional units. This chapter focuses on intracommunication in a computer system and the I/O techniques. The hardware and software aspects of communication between the subsystems of a computer are discussed in detail.

The main topics covered include various methods of performing I/O operations, the different types of I/O interfaces and I/O controllers and the common bus standards. The operational details of peripheral devices are discussed in Chapter 11.

 10.2 Accessing I/O Devices

Figure 10.1 illustrates general mechanism of accessing the I/O devices. An application program generally does not access the I/O devices directly. When it wants to perform an input or output operation, it makes a request to the operating system, by a OS CALL. The OS directs the request to the appropriate system software, the I/O driver, by transferring control to the I/O driver. The I/O drivers are collection of I/O routines for performing various types of I/O operations on different peripheral devices. The peripheral devices are linked to the CPU and the memory through the I/O controllers and the system bus. Each I/O device is interfaced to the bus through an I/O controller which takes care of physically interacting with the device and logically interacting with the I/O driver. The system bus is common to all the I/O controllers. Each I/O controller is allotted an address (or a set of addresses) for identification purposes. The operating system and I/O driver use this address for interacting with the device and performing an operation. The I/O driver indicates the required I/O operations to the I/O controller by issuing different commands. Each I/O controller can perform a set of commands. The I/O controller executes the command received, and issues required control signals to the I/O device. The I/O device takes appropriate action relevant to the command. This may involve input or output operation depending on the command. The system bus is a collection of three different groups of signals: address bus, data bus and control bus. The mechanism of accessing the I/O devices and performing various I/O operations are discussed in the following sections.



The sequence of events are indicated by numbers within brackets
The bus is common to all I/O controllers.

Fig. 10.1 Mechanism of accessing I/O devices

10.2.1 I/O Controller and I/O Driver

Every computer supports a variety of peripheral devices. For accessing a peripheral device in a computer, two modules are required:

- A *hardware module* called 'I/O controller' that interfaces the peripheral device to the system nucleus (CPU/memory) (see Fig 10.2)

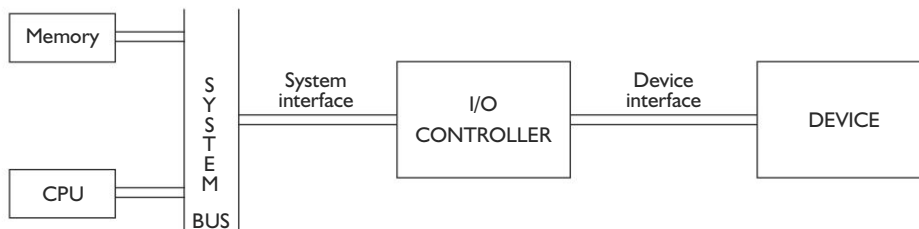


Fig. 10.2 I/O controller and device

- A *software module* called 'I/O driver' that issues different commands to the I/O controller, for executing various I/O operations, (refer Fig. 10.3).

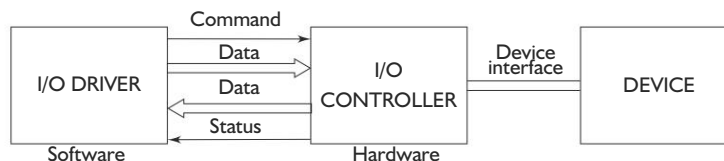


Fig. 10.3 I/O driver and device

10.2.2 Device Controller and Interfaces

A peripheral device is interfaced to the system nucleus, by a device controller, via the bus. Fig. 10.4 presents the block diagram showing the linkage of different hardware functional units to the bus. The bus consists of three groups of signals, each of which also form a bus: address bus, data bus and control bus. The data bus is used for three types of information exchange:

1. data between an I/O controller and processor
2. command from processor to the I/O controller
3. status from the I/O controller to the processor.

Each I/O controller is assigned a fixed set of addresses. To communicate with an I/O controller, the processor sends the address on the address bus. The addressed controller is logically connected to the bus, and all other controllers get logically disconnected from the bus.

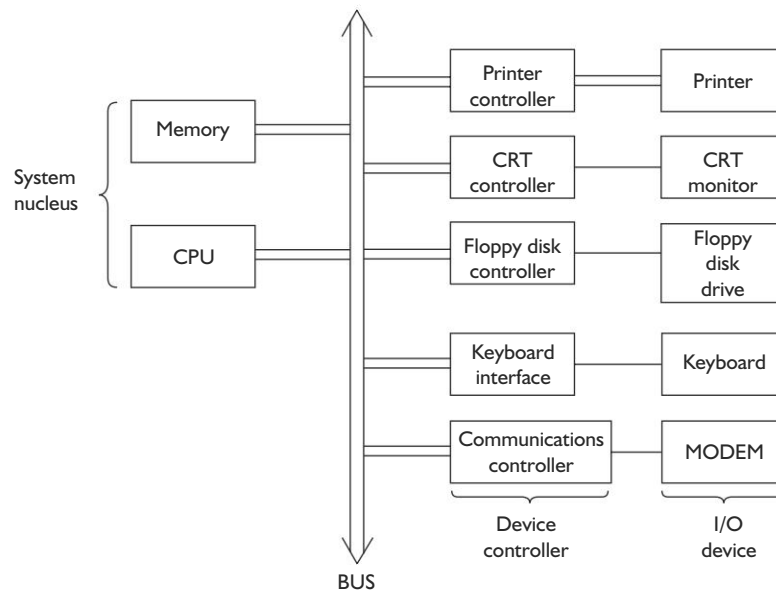
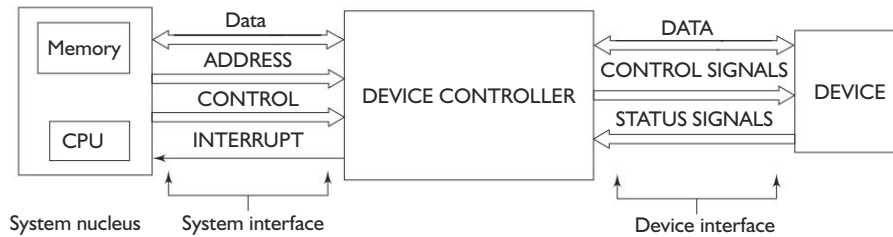


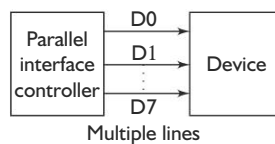
Fig. 10.4 Functional units and bus

The device controller is interfaced to the system bus on one side and to the peripheral device on the other side. These two sides are known as system interface and device interface respectively, as exhibited in Fig. 10.5.

**Fig. 10.5** Two ends of the device controller

The term ‘interface’ defines the signals between two subsystems and the protocol for communication between the subsystems. The system interface consists of signals between the system nucleus and the I/O controller. In a given system, the system interface for all I/O controllers is the same. The device interface consists of signals between I/O controller and I/O device. Device interface depends on the device type. The following are some examples of device interface: RS-232C, Centronics Interface, SA450, ST-506, SCSI, IDE, USB, FireWire.

There are two types of device interfaces: *Serial interface* and *Parallel interface*. In the Serial interface (Fig. 10.6), there is only one data line and the data bits in a byte are transmitted one after the other serially. The parallel interface (Fig. 10.7) provides eight data lines in parallel so that eight data bits (one byte) are transmitted from the system, simultaneously.

**Fig. 10.6** Serial interface**Fig. 10.7** Parallel interface

10.2.3 I/O Driver

The I/O drivers are programs that perform various I/O operations by issuing appropriate sequence of commands to the peripheral controllers and I/O devices. Following are certain operations performed by various I/O drivers:

- displaying a message on CRT
- printing some lines by the printer
- reading a file from a floppy diskette
- displaying the contents of a memory location
- saving (storing) the memory contents on a hard disk

Any I/O operation can be performed by calling the relevant I/O driver and passing relevant parameters for the operation. After completing the I/O operation, the I/O driver returns control to the called program and passes return parameters about the completion of the operation. Fig. 10.8 illustrates the three-tier interface between the device controller and the application program. The term 'BIOS' (Basic Input Output Control System) refers to the collection of I/O drivers, for different peripheral devices.

Hardware versus Software

The I/O driver for a given peripheral device can be developed if we know the architecture of the I/O controller. If the I/O controller is a dumb controller (e.g. printer controller) with limited capabilities, the I/O driver has to do several elementary functions. If

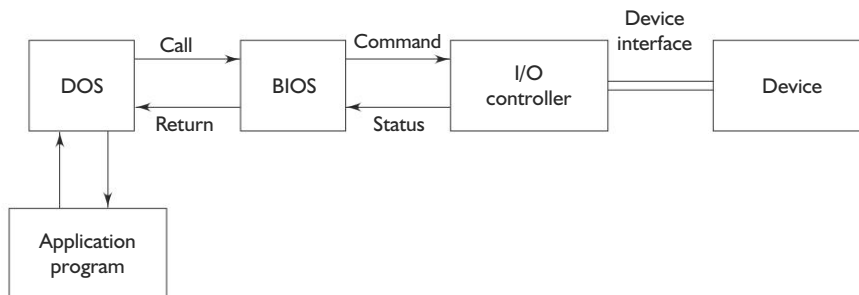


Fig. 10.8 Three-tier interface

the I/O controller is a sophisticated one (e.g. floppy disk controller), the I/O driver need to do only certain overall functions. The I/O driver software and the I/O controller hardware together achieve the I/O functions. In fact, the functions are shared between the hardware (I/O controller) and the software (I/O driver). The proportion of sharing is fixed by the architect/designer considering the various aspects such as cost and performance. When more functions are delegated to the I/O driver, the speed of operation is degraded but the hardware cost is reduced because there are less circuits in the I/O controller.

The I/O drivers for the basic peripheral devices in a personal computer are part of the BIOS, which is physically stored in ROM. The I/O drivers for other peripherals are usually given on CD. These are installed in the hard disk and brought into the RAM after booting.

10.2.4 I/O Ports

An I/O port is a program addressable hardware unit through which the CPU can transfer information. Different ports are allotted distinct addresses which are used while communicating with the ports. A port may be an independent hardware unit or it may be part of a hardware unit.

An input port puts data on the bus, whereas an output port accepts data from the bus. An input port gives data when it receives its address and IOR (I/O Read) signal as shown in Fig. 10.9. An output port receives data when it receives its address and IOW (I/O Write) signal as shown in Fig. 10.10. A port which sends data as well as accept data is known as an input/output port (I/O port) or bidirectional port, as shown in Fig. 10.11. In this case, there are two ports (an input port and an output port) with same address.

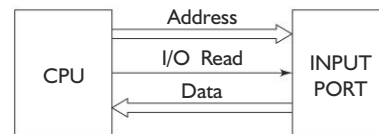


Fig. 10.9 Reading from an input port

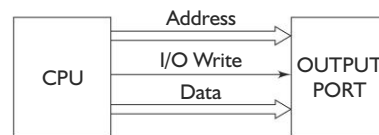


Fig. 10.10 Writing to an output port

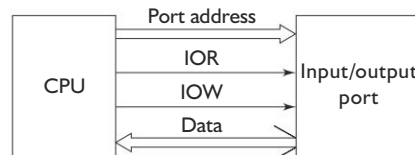
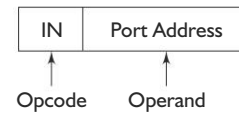


Fig. 10.11 Communication with a bi-directional port

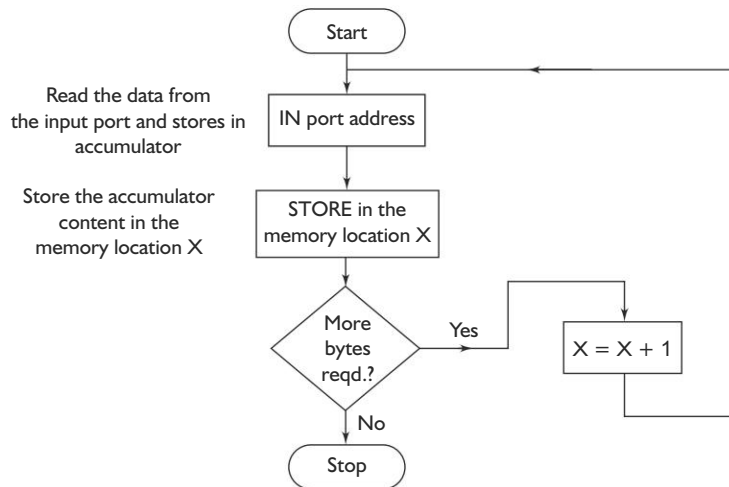
10.2.5 IN Instruction

A program reads the data from an input port by using an IN instruction whose format is presented in Fig. 10.12. The function of the IN instruction is transferring data from the input port to a CPU register. The actions performed by the CPU while executing the IN instruction are listed below:

- (a) The CPU sends the port address. Only the port whose address matches, gets selected.
- (b) The CPU transmits IOR signal. The port which is selected in step (a) responds to IOR signal and presents its data on the bus.
- (c) The CPU reads the data from the bus and loads the data in the processor register.

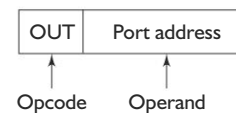
**Fig. 10.12** Format of IN instruction

The data in the register can be moved to memory by another instruction such as STORE or MOVE. The algorithm for reading a stream of data bytes from an input port is shown in Fig. 10.13.

**Fig. 10.13** Multiple byte reading from an input port

10.2.6 OUT Instruction

Assume that the data to be sent to the output port is in the memory location X. As a first step, this data is moved, from the memory location X, to the register in the CPU by Move or LOAD instruction. Next an OUT instruction is issued whose format is given in Fig. 10.14. The function of the out instruction is transferring the data from the register to the output port. The actions performed by the CPU for the OUT instruction are listed as follows:

**Fig. 10.14** Format of OUT instruction

- (a) The CPU sends port address. Only the port whose address matches, gets selected.
- (b) The CPU sends data (from the accumulator) on the bus.
- (c) The CPU sends IOW signal. The port which is selected in step (a) responds to IOW signal and procures data from the bus.

The algorithm for transmitting a stream of data bytes to an output port is shown in Fig. 10.15.

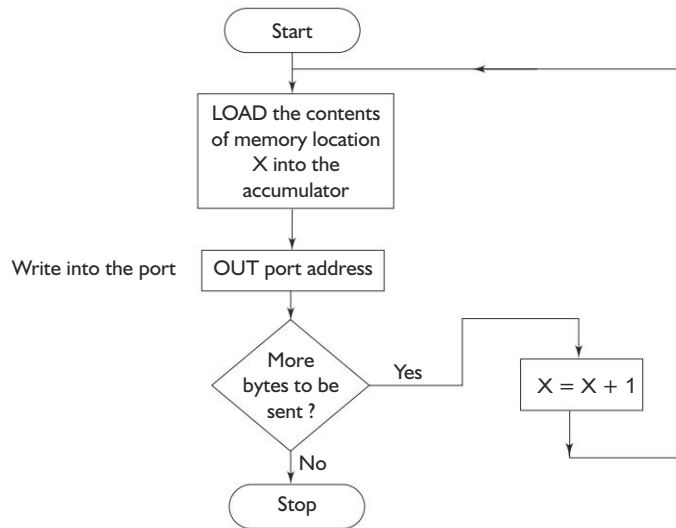


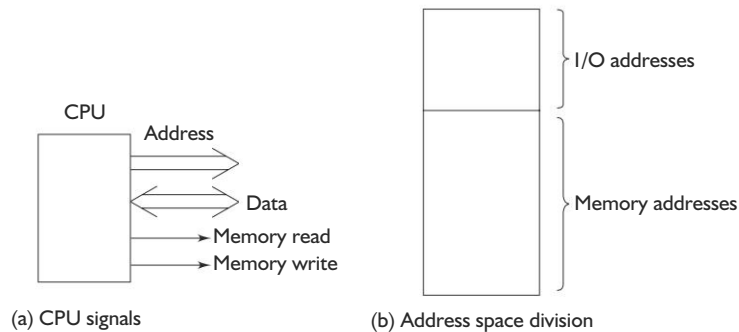
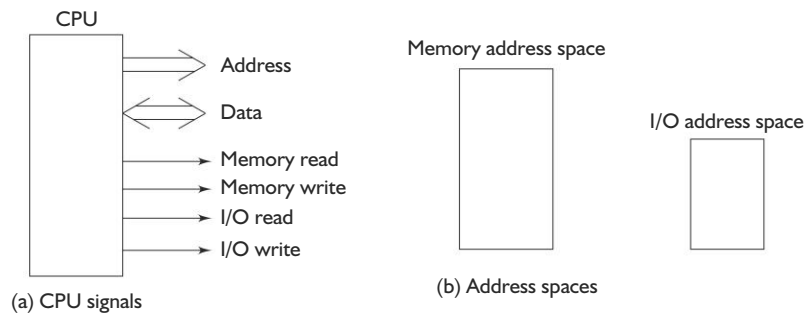
Fig. 10.15 Multiple byte writing to an output port

10.2.7 Memory Mapped I/O and I/O Mapped I/O

The type of I/O ports addressing discussed so far is known as the Direct I/O or I/O mapped I/O. In this scheme, the I/O ports are treated differently from memory locations and the program uses IN or OUT instruction for communication with ports. There is another scheme known as memory mapped I/O. In this scheme, the I/O ports are treated as memory locations. Hence, IN or OUT instructions are not used. The processor has no knowledge, whether it is accessing memory or I/O port. Only the program is aware of it. The total address space of the processor is distributed into I/O address space and memory address space. Hence, effective memory capacity is reduced by the amount of address space allocated to I/O ports. Table 10.1 provides a comparison between the two schemes. Figs. 10.16 and 10.17 exhibit the concept of memory mapped I/O and direct I/O. In a computer, any one scheme can be used.

TABLE 10.1 Memory mapped I/O and I/O mapped I/O

Sl. no.	Memory mapped I/O	I/O mapped I/O
1	Each port is treated as a memory location	Each port is treated as an independent unit
2	CPU's memory address space is divided between memory and I/O ports	Separate address spaces for memory and I/O ports
3	Extra circuit is needed for address space decoding to select memory or I/O ports	Extra signals (IOR and IOW) differentiate between memory and I/O ports
4	Single instruction can transfer data between memory and port	Two instructions are necessary to transfer data between memory and port
5	Data transfer is by means of instructions like MOVE	Each port can be accessed by means of IN or OUT instruction
6	Processor is not aware whether it is accessing a memory location or a port	Processor is aware whether it is accessing a memory location or a port


Fig. 10.16 Memory mapped I/O

Fig. 10.17 I/O mapped I/O

10.2.8 Device Controller Functions

Figure 10.18 illustrates the general organization of a device controller. The overall functions of a device controller are listed below:

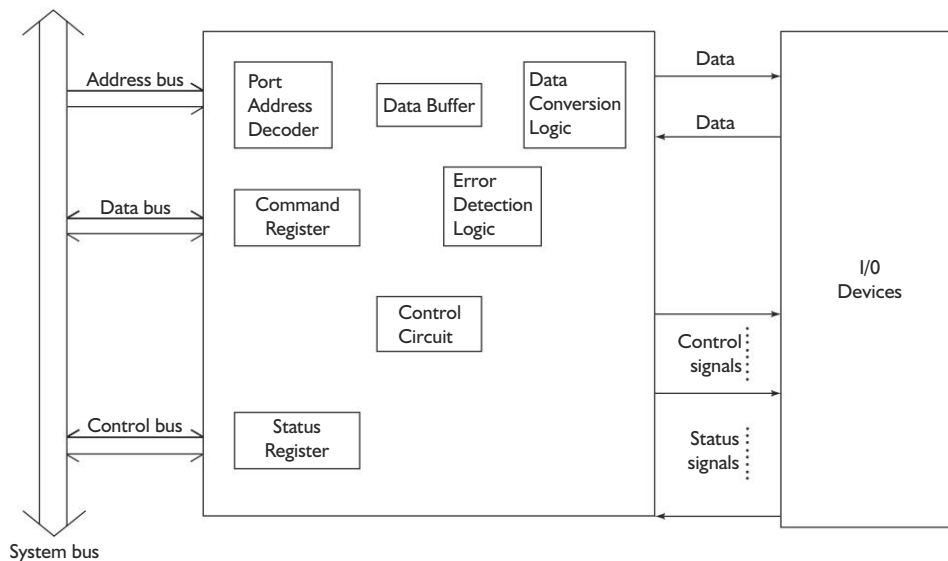


Fig. 10.18 Device controller organization

- (a) Receiving the command from the CPU.
- (b) Analyzing the command.
- (c) Issuing control signals to the device.
- (d) Receiving the status signals from the device and taking appropriate action.
- (e) Transferring data from CPU/memory to the device.
- (f) Transferring data from the device to the CPU/memory.
- (g) Converting the format of data received from the device e.g. serial to parallel.
- (h) Converting the format of data received from the CPU/memory e.g. parallel to serial.
- (i) Generating error checking code (parity bit, CRCC or ECC) during write operation.
- (j) Checking for error in the data received from the device.
- (k) Aborting the command execution on encountering any irrecoverable error.
- (l) Retrying the command on encountering an error.
- (m) Informing the CPU, the end of command execution by an interrupt.

A given device controller is designed to perform only some of these functions depending on the device and the system.

10.2.8.1 Command Types

The main function of the device controller is to execute a command transmitted by the CPU/software. The commands are of different types as follows:

- (a) Data transfer command
- (b) Status transfer command
- (c) Positioning control commands
- (d) Diagnostics command
- (e) Mode select command

Table 10.2 provides specific examples for different types of commands.

TABLE 10.2 Types of Commands

Device controller	Device command				
	Data transfer	Status transfer	Position control	Mode select	Diagnostic
Printer controller	Print	—	Skip line	Select programmed or interrupt mode	Loop back
Floppy disk controller	1. Read 2. Write	1. Sense drive Status 2. Sense Interrupt Status	1. Seek 2. Recalibrate	Specify FM or MFM mode	—
Hard disk controller	1. Read 2. Write	Sense status	1. Seek 2. Recalibrate		1. Test Controller 2. Test disk drive
CRT controller	—	—	Cursor position	1. Select graphics or text mode 2. Select black/white or color mode	—
Communication controller	1. Transmit data 2. Receive data	—	—	1. Select asynchronous or Synchronous communication	Loop back

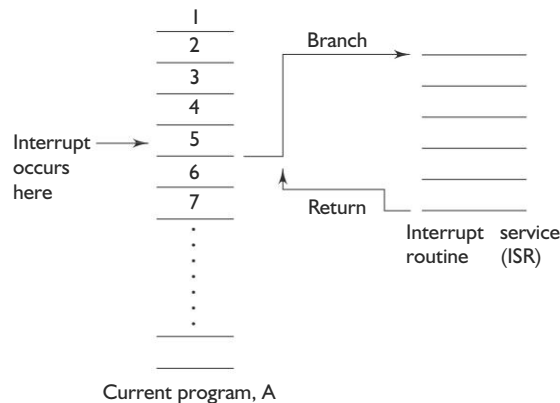


10.3 Interrupt Handling

Interrupt is a signal or event, inside a computer system, due to which the CPU temporarily suspends the current program execution and starts executing another program related to the interrupt. *Interrupt Handling* is a function performed jointly by hardware and software. In this section, essential aspects of interrupt handling is discussed.

10.3.1 Interrupt Concept

The CPU's activity at any instant is instruction processing. Except when the CPU is in HALT state, it is working (running) for some or other program. Interrupt is a signal or an event used to request the CPU to suspend the current program and take up another program related to the interrupt. In other words, on receiving an interrupt, the CPU should execute another program which will service the interrupt. This program is known as Interrupt Service Routine (ISR). Fig. 10.19 illustrates the interrupt concept. On completing the ISR, the CPU should usually, continue previously interrupted program, from the exact place where it left on receiving the interrupt. Hence before taking up ISR, the CPU stores the next instruction address (contents of program counter) at the time of branching to ISR.



Note : Interrupt occurs after commencement of the 4th instruction. The CPU recognises the interrupt before fetching the 5th instruction. The CPU suspends current program execution and branches to ISR. After completion of ISR, the CPU returns to the interrupted program, A. Now the 5th instruction is fetched.

Fig. 10.19 Interrupt concept

10.3.2 Interrupt Types

There are two type of interrupts: internal interrupts and external interrupts, Table 10.3 compares these two types.

TABLE 10.3 Internal and external

External interrupt	Internal interrupt
1. The external interrupts are generated by hardware circuits outside the CPU.	1. The internal interrupts are the interrupts generated within the CPU.
2. These are basically hardware interrupts.	2. These may be hardware or software interrupts.
3. The hardware interrupts are generated by the circuits.	3. The software interrupts are generated by an instruction in the program.

Fig. 10.20 lists the different types of interrupts. Definition of the different interrupt types are provided in the following paragraphs.

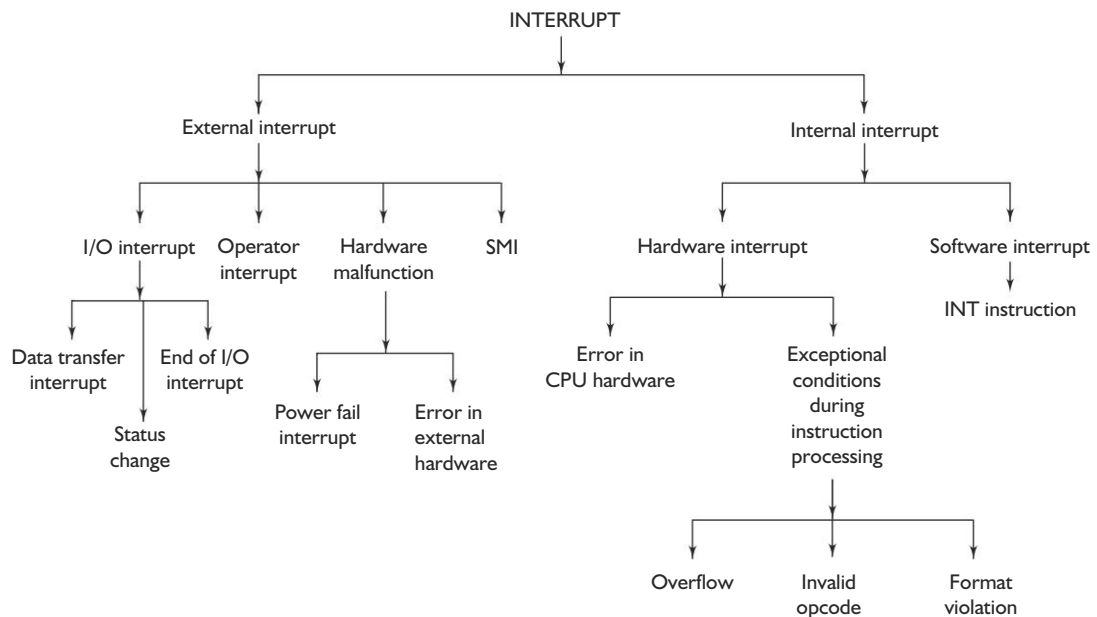


Fig. 10.20 Types of interrupts

10.3.2.1 Data Transfer Interrupt

The data transfer interrupt is raised by an I/O controller to indicate to the CPU that it is ready for data transfer. During the execution of a READ command, this interrupt is an indication of readiness of data with the device controller. On the other hand, during WRITE command, this interrupt is a demand for data by the device controller. Accordingly, the CPU program (ISR) should read one byte of data from the controller or issue one byte of data to the controller.

10.3.2.2 End of I/O Interrupt

The end of I/O interrupt is raised by an I/O controller on completing a command to indicate to the CPU (software) that the I/O operation is complete. This can imply any of the following two conditions:

- (a) The I/O operation is successfully completed.
- (b) The I/O operation is aborted due to some abnormal condition.

The ISR (I/O driver software) reads the status from the I/O controller and based on the status, the nature of completion of the I/O operation is confirmed.

10.3.2.3 Status Change Interrupt

This interrupt is used to inform the software when there is a change of status of an I/O device. For example, an I/O device may change from READY to NON-READY condition. In response, the device controller will generate this interrupt.

10.3.2.4 Power Fail Interrupt

This interrupt is raised by a special hardware which keeps monitoring the fluctuations in the AC power supply. This interrupt is an advance information that power is going to fail (cut-off) in a short while. The ISR can store the CPU status in some battery backed memory. Or it can initiate the operation of a standby power supply and the system can be used without disturbance for some time.

10.3.2.5 Operator Interrupt

This is a special interrupt to the operating system to perform a specific action.

10.3.2.6 External Hardware Malfunction Interrupt

When there is a hardware malfunction such as a memory failure or bus error, this interrupt is raised. Usually this will result in a NMI.

10.3.2.7 CPU Internal Hardware Error Interrupt

This interrupt indicates detection of some hardware malfunction within the CPU.

10.3.2.8 Program Exception Interrupts

These interrupts are caused due to abnormal conditions encountered by the CPU during the execution of the program. These are not due to hardware errors but due to special situations caused by programs. The various program exceptions are illegal opcode, instruction format violation, operand format violation and overflow. The ISR displays appropriate error message to the programmer.

10.3.2.9 Software Interrupt

The software interrupt is created by the program so that the CPU can temporarily branch from the current program to another program. INT (interrupt) instruction is used for generating this interrupt. Fig. 10.21 furnishes a format of INT instruction. The interrupt type number in the INT instruction is used to branch to the ISR.

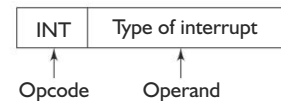


Fig. 10.21 Format of INT instruction

10.3.2.10 System Management Interrupt (SMI)

The SMI is a special purpose interrupt used for minimizing power consumption by powering off the peripheral subsystems which are not accessed for a long time. They will be powered on whenever accesses are made. This is a dynamic process in a system on continuous basis.

10.3.3 Interrupt Sensing

Instruction processing involves fetching and executing instructions in a program (Fig. 10.22). The CPU senses the presence of an interrupt after completing the execution

phase of the current instruction as indicated in Fig. 10.23. In case there is an interrupt, the CPU does not start the fetch phase of the next instruction. Instead, the CPU starts servicing the interrupt. Irrespective of when an interrupt occurs, the CPU usually services it only after completing the current instruction. Fig. 10.24 illustrates the effect of interruption.

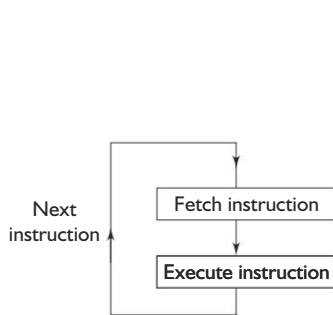


Fig. 10.22 Instruction cycle

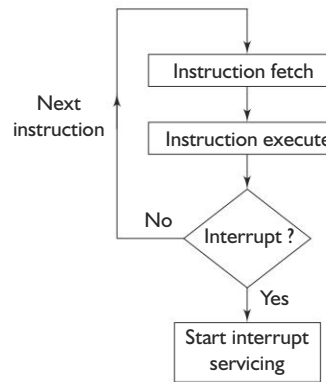
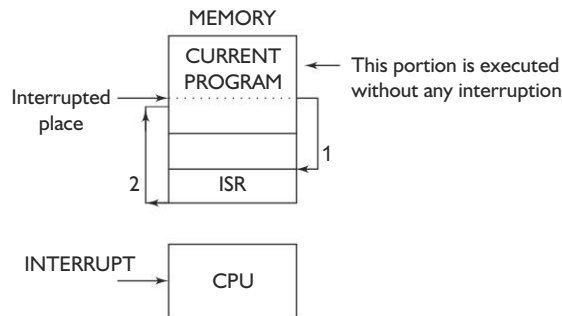


Fig. 10.23 Interrupt sensing by CPU



1. CPU branches to ISR on receiving interrupt
2. CPU returns to previous program on completion of ISR

Fig. 10.24 Interruption process

10.3.4 Interrupt Servicing

There are three distinct actions (Fig. 10.25) done automatically by the CPU hardware on sensing an interrupt. These are as follows:

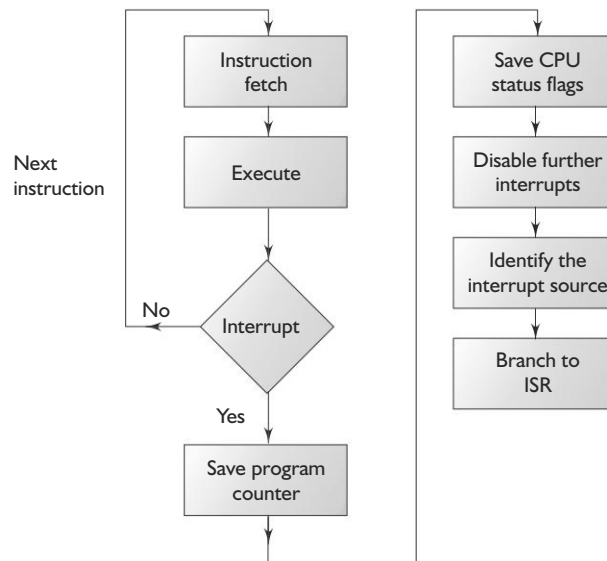


Fig. 10.25 Interrupt service sequence

- (a) Saving CPU status
- (b) Finding out the interrupt cause
- (c) Branching to the ISR

These actions are performed by the CPU hardware and no software is involved. The details are explained in the following sections.

10.3.4.1 Saving CPU Status

The interruption should not affect the net result or logic of the current program. Hence, the CPU stores the contents of the flags and the next instruction address (that is available in program counter) so that when it resumes this program, (after completing ISR), the same status can be restored back. Generally, a portion of read/write memory is used as stack to store the CPU status.

10.3.4.2 Identifying the Interrupt Source

In the PC, there is an interrupt controller, INTEL 8259, a support chip for the microprocessor INTEL 8088. It interfaces eight interrupt sources to the microprocessor as shown in Fig. 10.26. To find out the interrupt source, the CPU interrogates the interrupt controller. For this purpose, the CPU transmits interrupt acknowledgement (INTA) signal. The INTA signal conveys the following two messages to the interrupt controller:

- (a) The CPU has sensed the interrupt (INTR) signal issued by the interrupt controller.
- (b) The interrupt controller should now inform the CPU about which interrupt request should be serviced now, out of several requests.

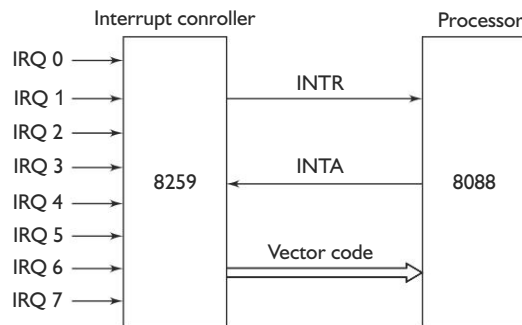


Fig. 10.26 Interrupt controller

In response to the INTA signal, the interrupt controller presents the ‘interrupt vector code’ to the CPU. The vector code identifies the interrupt request which should be serviced now by the CPU. Table 10.4 lists the eight different patterns of vector code and the corresponding interrupt requests. The five msbs are supplied to the interrupt controller by the software and these bits are indicated as x.

TABLE 10.4 Vector codes

Interrupt level	Vector code	Interrupt cause
0	xxxxx 000	IRQ0
1	xxxxx 001	IRQ1
2	xxxxx 010	IRQ2
3	xxxxx 011	IRQ3
4	xxxxx 100	IRQ4
5	xxxxx 101	IRQ5
6	xxxxx 110	IRQ6
7	xxxxx 111	IRQ7

10.3.4.3 Branching to ISR

Once the CPU identifies the source of interrupt, it should branch to the corresponding ISR. For this, the CPU should have a mechanism to find out the start addresses of different ISRs. A common method used is as follows: The ISRs of different interrupts are stored in different areas in memory. But, the start addresses of ISRs are stored in a fixed area in memory, one after the other, as a vector table as shown in Fig. 10.27. Depending on the vector code, the CPU accesses the corresponding entry in the vector table. The CPU fetches the ISR start address from the vector table, and then loads this address in the program counter. Thus, the ISR execution is commenced by the CPU.

The vector table is created by the system software. Each entry in the table contains four bytes as shown in Fig. 10.28. Two bytes presents the CS, segment start address of ISR, and the remaining two bytes furnishes the IP, offset for the ISR start address. The processor calculates the ISR start address by addition of these.

10.3.4.4 Returning from ISR

It is the responsibility of the ISR, to return the control back to the interrupted program, after the ISR is completed. A simple technique followed is using the IRET (return from interrupt) instruction in the ISR. By executing the IRET instruction, the CPU restores, from the stack, the CPU status corresponding to previously interrupted program. As a result, the CPU continues execution of the previously interrupted program. Hence, the main program will be executed as if *no interrupt* was encountered.

10.3.5 Interrupt Enabling and Disabling

If a program does not want any interruption, it can inform the CPU not to entertain any interrupt. A flag known as 'Interrupt Enable' (IE) is used for this purpose. When this flag is '1', the CPU responds to the presence of interrupt. When this flag is '0', the CPU ignores the

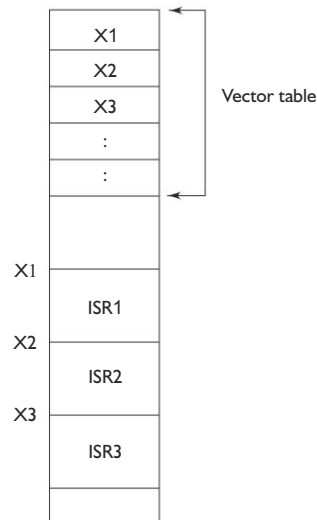


Fig. 10.27 Vector table and ISRs

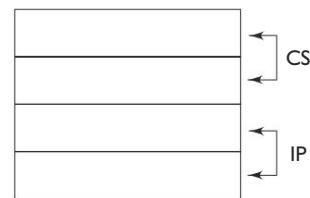


Fig. 10.28 Interrupt vector

presence of interrupt. Fig. 10.29 illustrates the modified instruction cycle, taking into account the interrupt enable flag. To set the IE flag, the program should issue 'Enable Interrupt' (EI) instruction. While executing this instruction, the CPU sets the IE flag. To reset the IE flag, the program should issue 'Disable Interrupt' (DI) instruction¹. While executing this instruction, the CPU resets the IE flag. Thus the CPU's behavior, as regard to interrupt servicing, is controlled by the program that is being executed currently. There are two special situations when the CPU resets the IE flag on its own as listed below:

1. During reset sequence (power-on or manual reset), the CPU clears the IE flag.
2. During interrupt servicing, the CPU resets the IE flag immediately after saving the CPU status and before branching to ISR. The objective is this: when CPU starts execution of ISR, interrupts are disabled. Subsequently, it is up to the ISR to allow interrupts (by an EI instruction) if it does not mind interruption.

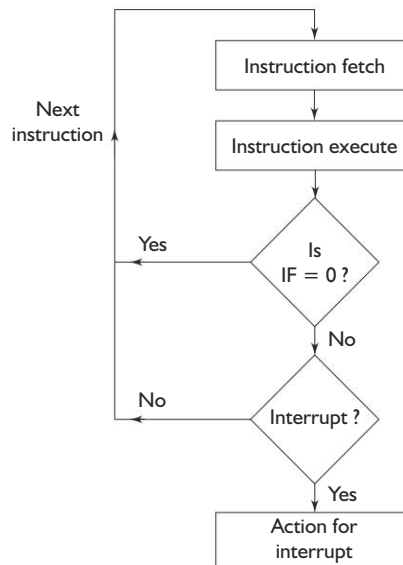


Fig. 10.29 Interrupt sensing and interrupt enable flag

10.3.6 Interrupt Nesting

When the CPU is executing an interrupt service routine, if it allows another interrupt, it is known as interrupt nesting. Suppose initially, the CPU is executing program A when inter-

¹ The STI and CLI instructions are similar to EI and DI instructions, respectively.

rupt 1 occurs. The CPU after performing interrupt service actions (saving CPU status, identifying interrupt source and branching to ISR) starts executing ISR1. Now, let us say interrupt 2 occurs. The CPU again performs interrupt service actions and starts executing ISR2. When it is executing ISR2, interrupt 3 occurs. The CPU again performs interrupt servicing and starts executing ISR3. After completing ISR3, the CPU returns and continues the remaining portion of ISR2. After completing the ISR2, the CPU executes the remaining portion of ISR1. After completing ISR1, the CPU returns to the program A and continues from the place it branched earlier. The main program, A, is not aware of the interrupts 2 and 3. It is suspended due to interrupt 1 and resumed when ISR1 is completed.

10.3.7 Interrupt Priority

The interrupt requests from various sources are connected as input to the interrupt controller. As soon as the interrupt controller senses the presence of any one or more interrupt requests, it issues an interrupt signal to the CPU. The interrupt controller assigns a fixed priority for the various interrupt requests. For instance, the IRQ0 is assigned the highest priority among the eight different interrupt requests. Assigning decreasing order of priority from IRQ0 to IRQ7, the IRQ7 has the lowest priority. It is serviced only when no other interrupt request is present.

10.3.8 Selective Masking

Suppose the program allows only certain interrupts but wants to disable other interrupts. This is achieved in two steps. First, the program issues EI instruction so that the IE flag is set and the CPU does not ignore the interrupt. Second, the program directs the interrupt controller to ignore certain interrupt request inputs. Now on, the interrupt controller ignores interrupt requests at these inputs. Thus, the program has selectively masked certain interrupts. The mask pattern presented by the program, directly to interrupt controller, decides which interrupts are masked at any time. Certain interrupts may be of higher urgent nature than the current program and hence not masked. Other interrupts of lower urgency than the current program are masked so that the current program is executed without too many interruptions. Suppose two interrupts are present at a given time. By masking a higher priority interrupt at the interrupt controller level, a lower priority interrupt is serviced first whereas the higher priority interrupt is kept pending.

10.3.9 Non-maskable Interrupt

Non-Maskable Interrupt (NMI) is raised for a cause that needs urgent attention of the CPU (ISR). The processor attends to NMI as soon as it occurs. Sensing of NMI does not depend on IE flag. The processor is designed with a fixed vector code for NMI. The 8088 microprocessor assumes a vector code of 2 and accesses the corresponding entry in the vector table. Some common examples for NMI causes are as follows:

1. Hardware error
2. Power fail interrupt

10.3.10 CASE STUDY 1: Programmable Interrupt Controller (PIC) 8259A and PC

The PIC is used in a PC to handle maskable interrupts. The 8259A IC services upto 8 interrupts. Its functions are grouped into four types.

1. Sense the interrupt requests and raise the interrupt signal.
2. Determine priority when multiple interrupt requests are present.
3. Present vector code, corresponding to the highest priority interrupt request.
4. Ignore certain interrupt requests according to the interrupt mask pattern.

The 8259A has two different modes of operation based on processor used: (a) 8080/8085 mode and (b) 8086/8088 mode. The desired mode of operation is indicated to 8259A in a command word supplied by the processor. There are some more parameters to be supplied to 8259A by the program.

1. ICW (Initialisation Command Word).
2. OCW (Operation Command Word).

Up to four ICWs are issued, one after another, by the program. The different details provided through ICWs are:

1. Whether one 8259A is present or multiple 8259As are cascaded.
2. Whether interrupt requests are level triggered or edge triggered inputs.
3. Interrupt service routine's page start address.
4. Whether an input is from a slave 8259A or from an interrupt request.
5. The identification code for each slave 8259A.
6. Whether buffered mode or fully nested mode is required.
7. Whether 8080/8085 mode is desired or 8086/8088 mode is desired.

In a PC, there is a single 8259A. Hence the 8259A is operated in a simple configuration. The BIOS (and POST) firmware supply appropriate ICWs to the 8259A through a series of OUT instructions.

The OCWs command the 8259A to operate in one of the four interrupt modes namely fully nested mode, rotating priority mode, special mask mode and polled mode.

The OCWs should be supplied to the 8259A only after issuing ICWs. The different details supplied by the OCWs are

1. Interrupt mask which informs the 8259A the interrupts that should be enabled and those which should be masked
2. Control bits for 'rotate' and 'end of interrupt' modes
3. Normal mask mode or special mask mode
4. Polling

The overall block diagram is shown in Fig. 10.30.

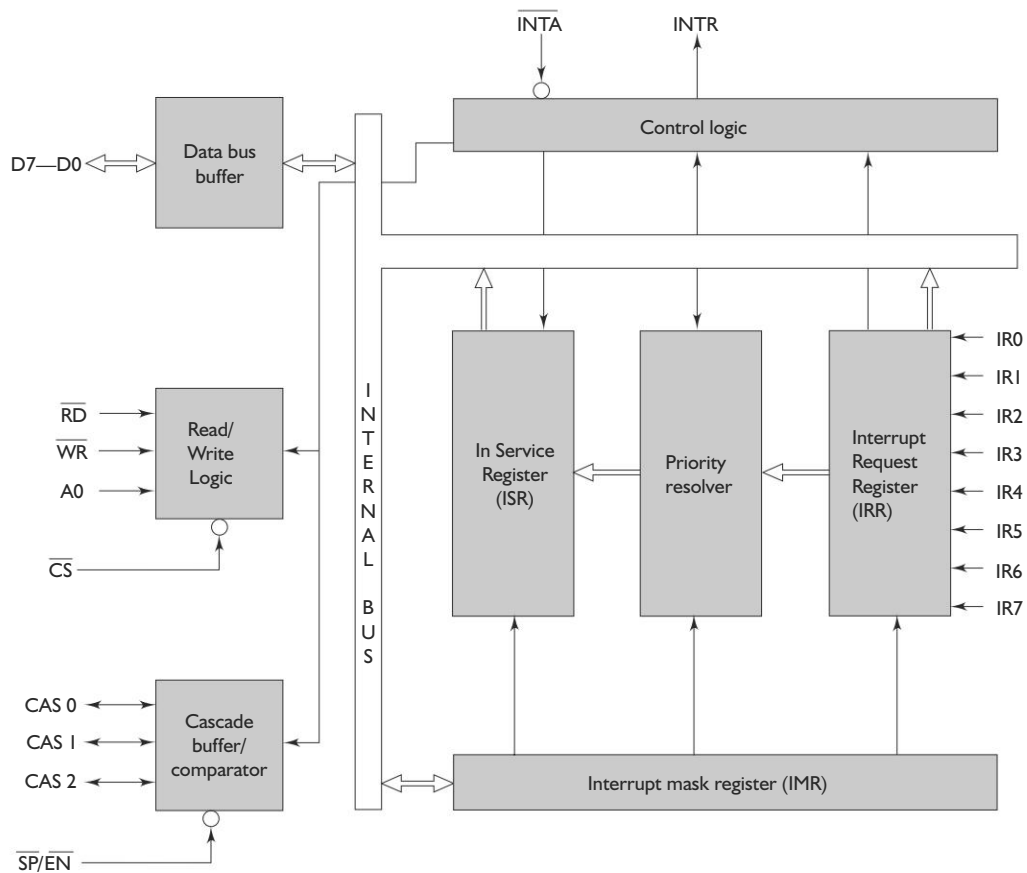


Fig. 10.30 8259A Block diagram

10.3.10.1 Interrupt Sequence in PC

The firmware initialises the 8259A and makes it operate in the 8088 mode as a single PIC with no cascading. The interrupt requests are edge triggered. The firmware also informs the start address of ISRs. Only five address bits A15–A11 are supplied by the firmware. The remaining bits are formed by the 8259A and 8088 jointly. Once programmed properly, the 8259A is ready to service interrupt requests. The interrupt sequence is described below.

1. When any interrupt request input is high, the corresponding bit in the IRR is set.
2. When one or more IRR bits, which are not masked by the corresponding IMR bits, is present, the 8259A sends an INTR signal to the 8088.
3. The 8088 recognises the INTR signal before fetching a new instruction if IE is set (i.e., interrupt is enabled by software). It stores 8088 status in stack and resets IE.
4. The 8088 starts the first interrupt acknowledgement cycle. The $\overline{\text{INTA}}$ signal is made low. The 8088 also makes $\overline{\text{LOCK}}$ signal low.
5. The 8259A on receiving $\overline{\text{INTA}}$, sets the ISR bit corresponding to the highest priority IRR bit. This IRR bit is reset immediately. Now, the 8259A interrupt logic is frozen.
6. The 8088 removes $\overline{\text{INTA}}$ in T4.
7. After completion of the first interrupt acknowledgement cycle, the 8088 again repeats the same cycle without any break. The $\overline{\text{LOCK}}$ is not removed in the end of the first cycle. It again makes $\overline{\text{INTA}}$ low.
8. On receiving the second $\overline{\text{INTA}}$, the 8259A issues an 8 bit vector code on the data bus. The vector code pattern is shown in Fig. 10.31.

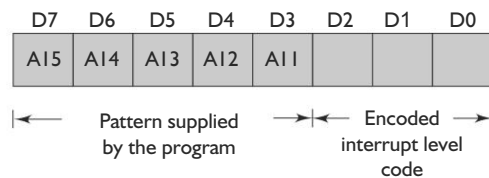


Fig. 10.31 Vector code pattern

9. The 8088 multiplies the vector code by 4 and uses this as the vector number.
10. The 8088 fetches 4 bytes from the memory location given by the vector number. These 4 bytes point to the beginning of the ISR.
11. The 8088 loads 2 bytes in CS and 2 bytes in IP and starts instruction fetching and execution of the ISR. The CS contains code segment (ISR) start address, and the IP (instruction pointer) contains the offset address.

Figure 10.32 illustrates the sequence of maskable interrupt servicing by 8088.

I/O Ports in 8259A

There are multiple registers within the 8259A. Logically, the 8259A appears to have two input ports and two output ports.

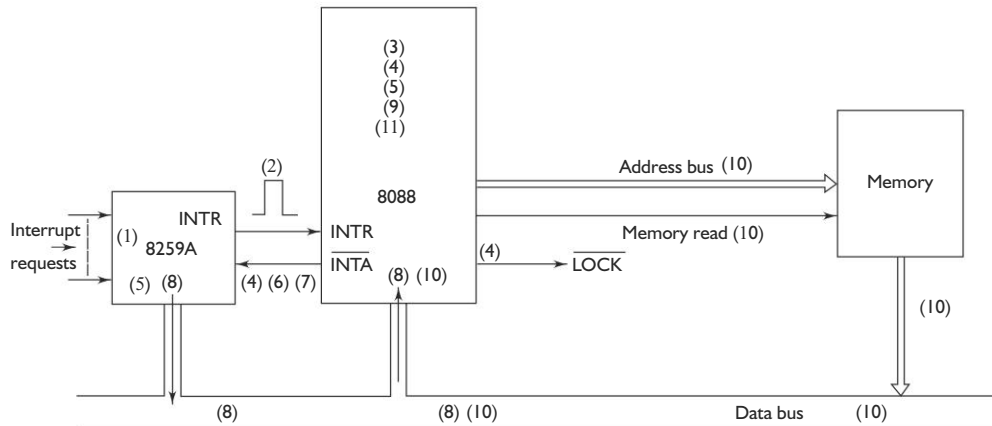


Fig. 10.32 Maskable interrupt servicing by 8088

Maskable Interrupts in PC

In 8088-PC, eight maskable interrupts are used for various purposes listed in Table 10.5.

TABLE 10.5 Interrupts in IBM PC

Level	Signal	Source	Remarks
0	IRQ0	Timer 0 Tick	Generated at the rate of 18.2 cycles/sec. Used to maintain and update date and time.
1	IRQ1	Keyboard interface	On receiving a 'scan code' from the keyboard, the keyboard interface generates IRQ1.
2	IRQ2	—	Not used in common PCs. Used as interrupt from special subsystems, e.g. scanner
3	IRQ3	Serial port (Secondary)	Generated for every byte of data to be transferred; also for error condition
4	IRQ4	Serial Port (Primary)	Generated for every byte of data to be transferred; also for error condition
5	IRQ5	Hard disk Controller	Generated at the end of data transfer operation or completion of a command by the HDC
6	IRQ6	Floppy disk Controller	Generated at the end of data transfer operation or completion of a command by the FDC
7	IRQ7	Parallel Port	Generated for every byte of data transferred

10.4 I/O Techniques in PC

The term 'Data Transfer' refers to the moving of information, between the CPU/memory on one side, and the I/O devices/peripherals on the other side. The data transfer is done usually in steps of one word at a time. The destination of any information presented to the computer is ultimately the main memory. Similarly the source of information received from the computer is the main memory. This section discusses the concepts and techniques followed by different methods of performing I/O operations.

10.4.1 Hardware and Software Methods

The following are the two types of data transfer as shown in Fig. 10.33:

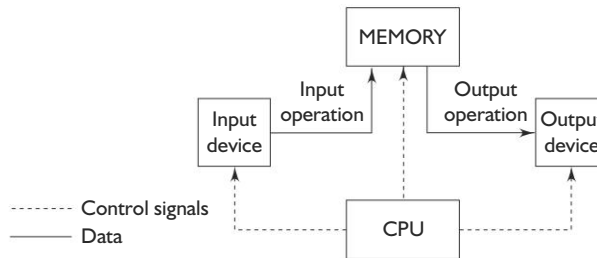


Fig. 10.33 Two types of data transfer

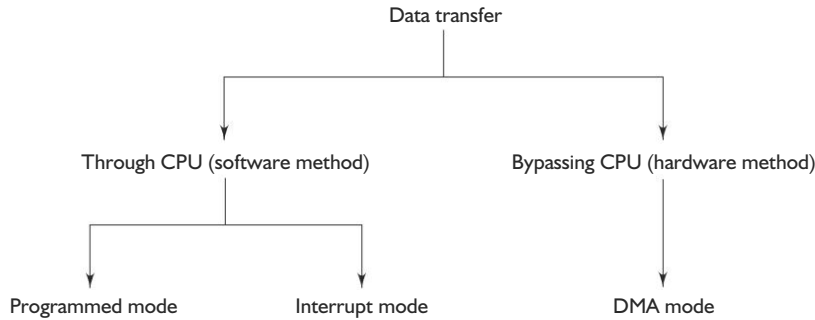
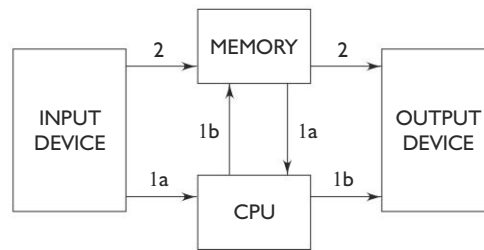
- (a) From an input device to the memory.
- (b) From the memory to an output device.

There are two different techniques of performing data transfer:

- (a) Passing the data through the CPU (Software method).
- (b) Bypassing the CPU (Hardware method).

This is illustrated in Fig. 10.34. The main difference between the *Software method* and the *Hardware method* is the extent of involvement of CPU, in the actual data transfer operation. In the software method, the tasks related to input/output operation are implemented as a program/routine which is executed by the CPU. Hence, the CPU is the total incharge of the I/O operations. In the hardware method, the program delegates the responsibility of performing I/O operations to another hardware unit, called DMA Controller.

As shown in Fig. 10.35, there are two steps in the software method. For example, to move a byte of data from an input device, the following two steps are performed:

**Fig. 10.34** Techniques of data transfer**Fig. 10.35** Principles of data transfer

- (a) Read the data byte from input device to the CPU (Step 1a).
- (b) Move the data byte from the CPU to the memory location (Step 1b).

These two steps are repeated to transfer every byte of data. Each of these steps is achieved by programming the CPU, i.e. by using appropriate instructions. Hence, this method is termed as software method. In the hardware method, the CPU software is not involved in actual transfer of data bytes. The software only provides certain parameters initially to the hardware, and the hardware performs the actual transfer of data bytes without involving CPU. This method is popularly known as DMA (Direct Memory Access) technique.

The software method of data transfer is of two types:

- (a) Programmed mode (Polling)
- (b) Interrupt mode

The programmed mode of data transfer is a dedicated task performed at a stretch (from the beginning of first byte till the end of the last byte) by the software. In the interrupt mode, the software (CPU) is invoked by the hardware only during the actual data transfer of every byte, and at other times, it is free to do any other task. In the DMA mode, the software is not at all involved in data transfer. Instead, the DMA controller or data channel takes care of

transferring data between memory and I/O device. The software's responsibility in DMA method, is issuing required information about the I/O operation to the DMA controller. Once this is done, the DMA controller takes care of data transfer between memory and I/O device, without the involvement of CPU. After the I/O operation is complete, the CPU/program is informed about the completion.

10.4.2 Programmed Mode or Polling

In the programmed mode, the program or I/O routine performs four distinct activities for each and every data byte transferred:

- (a) Reading the status of the peripheral device
- (b) Analyzing whether the device is ready for data transfer or not
- (c) If the device is ready, going to step (d) for actual data transfer; if the device is not ready, going to step (a) in order to loop on till the device is ready for data transfer
- (d) Performing the data transfer in two steps. For an input operation, the two steps are as follows:
 - (i) transferring the data from the input device to the CPU
 - (ii) storing the data in a memory location

For an output operation, the two steps are as follows:

- (i) Loading the data from the memory to the CPU
- (ii) Issuing the data to the output device

Fig. 10.36 presents a block diagram for programmed mode for print operation. Fig. 10.37 presents a flow chart for programmed mode data transfer.

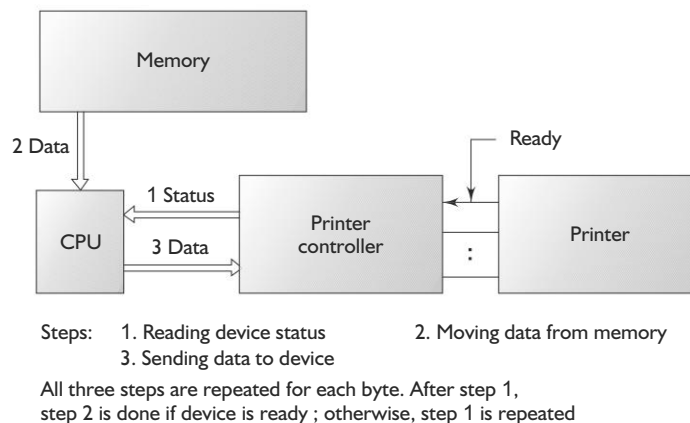


Fig. 10.36 Programmed mode block diagram—print operation

10.4.2.1 Drawback in Programmed Mode

It is easily observed that the speed of data transfer in programmed mode depends on the number of times, the steps a to c are repeated; this in turn depends on the speed of the

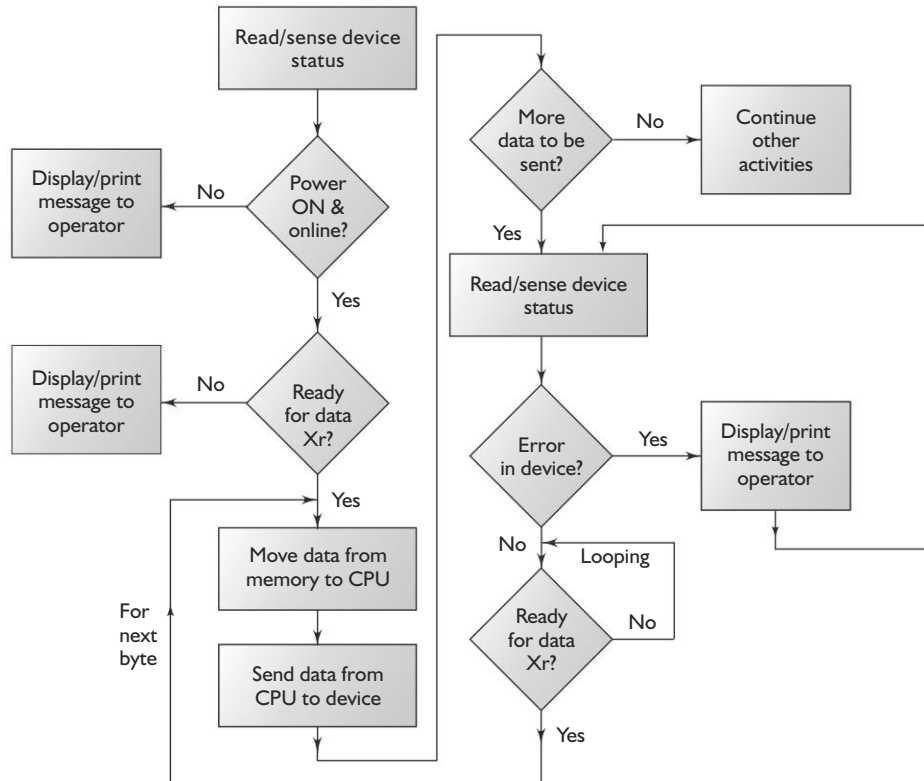


Fig. 10.37 Programmed mode of data transfer (output operation)

peripheral device. If the device is slow, the I/O routine loops on these three steps for a long time. In other words, CPU's time is wasted.

10.4.3 Interrupt Mode

In the programmed mode, the device status is monitored by the software (I/O routine). Whereas, in the interrupt mode, the software (I/O routine) does not wait until the device is ready. Instead, the device controller continuously monitors the device status, and raises an interrupt to the CPU, as soon as the device is ready for data transfer. The I/O routine

software gets hold of the CPU immediately. After completing one byte of data transfer, the I/O routine releases the CPU which is free to perform any other program or routine. When the next interrupt is generated, again the I/O routine gains control. Thus, in the interrupt mode, the software (CPU) performs data transfer but is not involved in checking whether the device is ready for data transfer or not. In other words, the steps a to c of programmed mode, are delegated to device controller hardware. This, obviously, leads to better utilization of CPU. The CPU can execute some other program until the interrupt is received from the device. To operate in interrupt mode of data transfer, the device controller should have additional intelligence for checking device status and raising an interrupt whenever data transfer is required. This results in extra hardware circuitry in the controller. Figs. 10.38 and 10.39 illustrate the interrupt mode of data transfer.

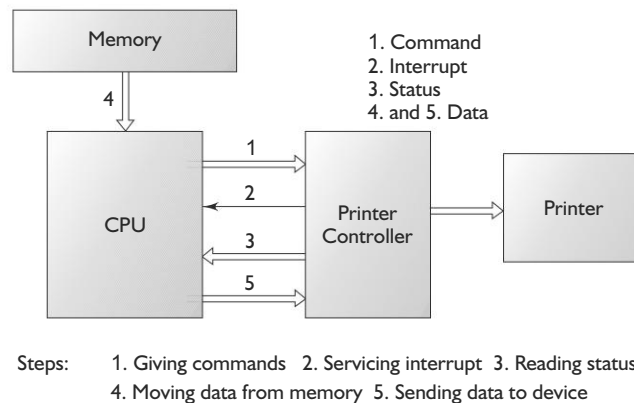
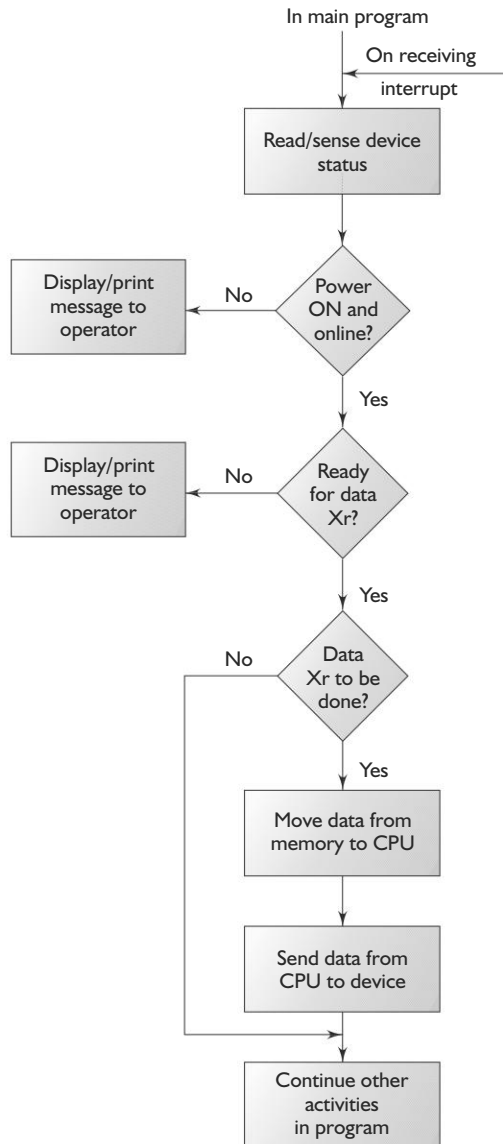


Fig. 10.38 Interrupt mode block diagram

10.4.4 DMA Mode of Data Transfer

In DMA mode, the software performs only 'initiation' which involves delivering commands to the DMA controller and the device controller. The actual operations related to transfer of the data bytes is performed by the DMA controller which is an independent hardware unit. The DMA controller can access memory, for read or write operation, without any assistance from the CPU. The device controller requests the DMA controller that one byte is to be transferred (between the memory and the device controller), instead of interrupting the CPU.

**Fig. 10.39** Interrupt mode of data transfer

The software provides the following DMA parameters to the DMA controller:

- (a) Memory Start Address
- (b) Byte Count
- (c) Direction: Input or Output

The 'Memory Start Address' specifies the memory location in which the first data byte is stored or read. The 'Byte Count' specifies the number of bytes to be transferred. The 'Direction' specifies whether the data transfer is an input or output operation. An input operation involves accepting data from the device controller, and writing it in the memory. An output operation involves reading data from the memory, and supplying it to the device controller.

Apart from issuing DMA parameters to the DMA controller, the software also issues a command and relevant command parameters to the device controller. For example, to read data from the floppy diskette, 'Read' command is issued to floppy disk controller along with the relevant command parameters such as the Track number, Side (head) number, Starting Sector number and Ending Sector number. Figs. 10.40 and 10.41 present the DMA mode of data transfer.

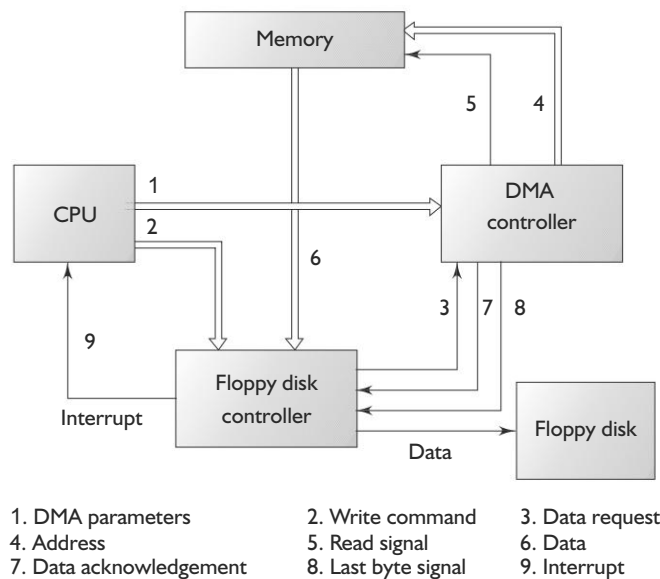
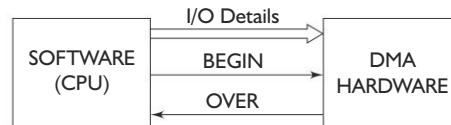
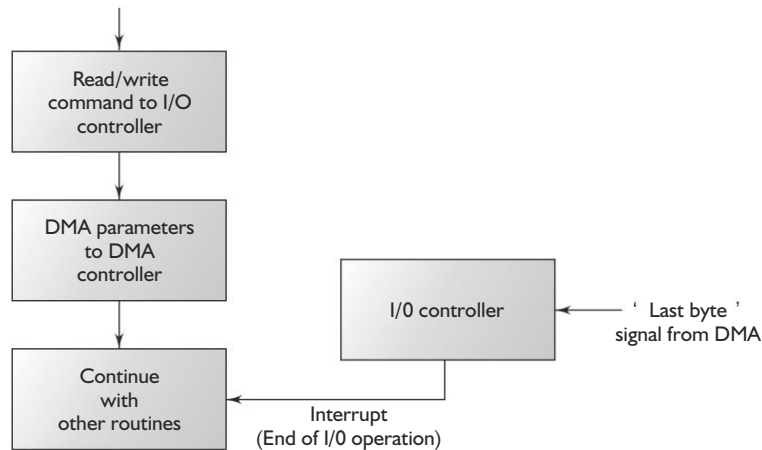


Fig. 10.40 DMA mode block diagram—Floppy disk write

The DMA controller has registers in which it stores the DMA parameters offered by the software. Once the DMA parameters are received, the DMA controller is ready for data transfer, but, the initiative should come from the device controller.

**Fig. 10.41(a)** Overall protocol for DMA**Fig. 10.41(b)** DMA mode data transfer – CPU/program actions

10.4.4.1 DMA Sequence

Consider the example of reading one sector of data from floppy diskette. Let us assume that we are interested in reading only initial 400 bytes of data from sector number 2 on track number 5 of side 1. The I/O routine issues the following parameters to the DMA controller:

- (a) Start Address: 60000
- (b) Byte Count: 400
- (c) Direction: Input

The I/O routine also issues the following command and command parameters to the floppy disk controller:

- (a) READ DATA command
- (b) Track number: 5
- (c) Side number: 1
- (d) Starting sector number: 2
- (e) Ending sector number: 2

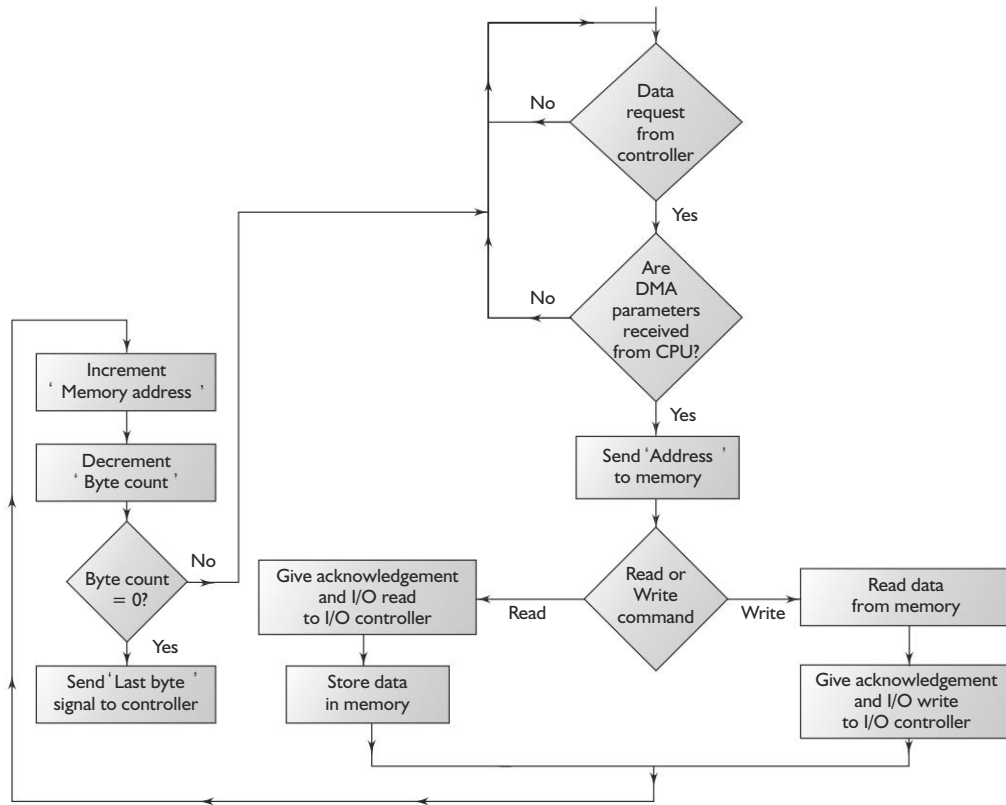


Fig. 10.41(c) DMA mode data transfer—DMA controller actions

On receiving the READ DATA command, the Floppy Disk Controller (FDC) selects the head 1, and it starts reading data bit stream from the floppy diskette. The floppy disk controller starts assembling the data bits into bytes (i.e. serial to parallel conversion). The data on sector 2 has to be transferred to memory. The FDC identifies the sector 2 from the ID field of sector 2. The ID field is not transferred to memory. As soon as one byte is ready, the controller raises DMAREQ (DMA Request) signal, which is transmitted to the DMA controller. This indicates that one byte has to be transferred. Immediately, the DMA controller raises HOLDREQ (Hold Request) signal which reaches the CPU. This indicates that the DMA controller needs the bus control in order to transfer the data byte. If the CPU is not using the bus, it issues sanction to DMA controller by raising the signal HOLDACK (Hold Acknowledgement). Now, the DMA controller takes over the bus. Immediately, the DMA controller issues DMAACK (DMA Acknowledgement) signal acknowledging the DMAREQ from the floppy disk controller. In response, the FDC places the data byte on the bus. At the same time, the DMA controller issues the memory start address and MEMW

(Memory Write) signal to the memory. Now, the memory stores the data present on the data bus, into the location identified by address bus. Thus, one byte has been transferred from floppy disk controller to memory. At the same time, the DMA controller performs two house-keeping operations listed below:

- (a) Increments the memory address
- (b) Decrements the byte count

The FDC drops the DMAREQ signal. In response, the DMA controller drops the DMAACK signal. It also drops HOLDREQ signal. In response, the CPU withdraws the bus sanction to DMA controller by dropping HOLDACK signal.

The above DMA sequence is repeated for every byte. However, the DMA controller should not transfer more than the 'byte count' parameter. Hence, when the byte count reaches zero, the DMA controller informs the FDC, not to attempt for further data transfer. For this purpose, the DMA controller issues the LAST BYTE (TERMINAL COUNT) signal along with the DMAACK signal. Now, the FDC understands that this is the last byte being transferred and *no more* data transfer request is be raised by the FDC.

During the entire DMA sequence, the software (and the CPU) is unaware of the progress of this particular I/O operation. On receiving LAST BYTE signal, the FDC raises an interrupt to the CPU. Now, the CPU switches to the software which initiated the I/O operation. Thus, the interrupt from the device controller is an information to the software that the I/O operation is complete.

Honest method and cycle stealing

There are two ways of obtaining bus control, to the DMA controller, to perform a data transfer. In one method, the HOLD REQ signal, from the DMA controller, is directly provided to the CPU which releases the bus control, and issues sanction by generating HOLDACK signal (refer Fig. 10.42). This is a straight-forward request/sanction protocol. In this case, the CPU knows that someone else (DMA Controller) is using the bus. Some

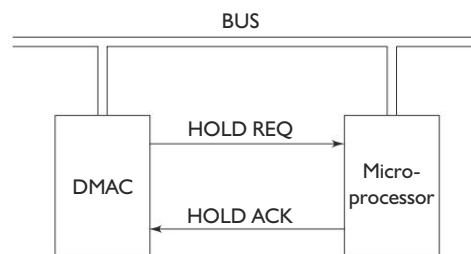


Fig. 10.42 Honest method of bus sanction

systems follow an indirect technique known as “cycle stealing”. The HOLDREQ signal is not directly transmitted to the CPU; instead, it is issued to another circuit known as Bus Arbitration Logic (BAL) (refer Fig. 10.43). The BAL constantly monitors the bus activities of the CPU. Whenever the CPU does not need the bus, the BAL issues sanction to the DMA controller by sending the signal HOLDACK. At the same time, the BAL performs two more actions, to prevent the CPU from using the bus until the DMA controller surrenders the bus:

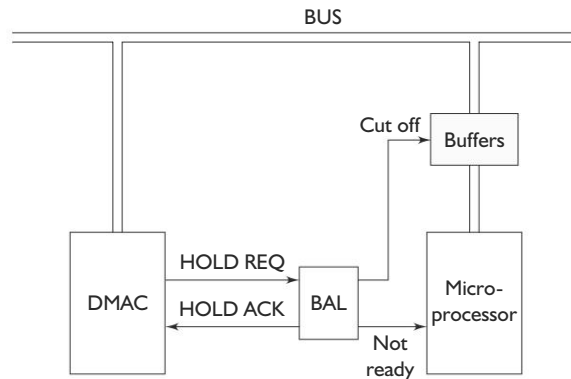


Fig. 10.43 Cycle stealing method

- (a) Cutting off the CPU signals from the bus.
- (b) Signalling *NOT READY* status to the CPU.

The first action safeguards that the DMA controller’s bus sequence, is not disturbed. The second action ensures that the CPU cycle is frozen, by forcing *WAIT STATE* in case it starts a bus activity. In fact, this is a false wait state, since the CPU is cheated by the BAL. The CPU assumes that the wait state is a genuine wait state caused by a slow memory or I/O device accessed by it.

10.4.4.2 Advantages of DMA Mode

It is obvious that DMA mode is a hardware technique of data transfer. It has two distinct advantages over the programmed mode and interrupt mode as follows:

- (a) Supporting high speed peripheral devices
- (b) Achieving parallelism between CPU processing and I/O operation

High Speed I/O Devices

In the case of high speed I/O devices such as hard disk and magnetic tape, time taken for one byte of data transfer should be as short as possible to avoid loss of data. For instance, a hard disk rotates at a high speed such as 7,200 rpm. Hence, while reading data, the hard disk controller generates data transfer requests at a very high rate. Handling such a high

rate is not a problem in DMA mode since the data transfer sequence in DMA is hardware controlled, and involves only performing memory write or memory read operation.

Parallelism between CPU and DMA Controller

Since DMA controller manages data transfer between memory and I/O devices, the CPU can do some processing activity during the transfer of a stream of data. For this purpose, the program is structured such that the CPU does not idle during data transfer. In a multi-programming system, the CPU can do processing for one program, while the DMA controller is performing an I/O operation for another program. In other words, two or more programs are run concurrently, with the CPU and DMA controller operating simultaneously for different programs.

Table 10.6 compares the three modes of data transfer techniques.

TABLE 10.6 Comparison of data transfer techniques

Sl. no.	Function	Programmed mode	Interrupt mode	DMA mode
1	Checking device ready status before each byte transfer	I/O routine software; loops on	Device controller hardware; raises interrupt	Device controller hardware; raises DMA request
2	Reading data from or issuing data to device controller	INPUT instruction or OUTPUT instruction in the I/O routine	INPUT instruction or OUTPUT instruction in the I/O routine	DMAACK signal from DMA controller along with IOR/IOW
3	Storing data in memory or reading data from memory	Move or load instruction in the I/O routine	Move or load instruction in the I/O routine	Memory write or read operation by DMA controller
4	Generating address of memory location	I/O routine	I/O routine	DMA controller
5	Maintaining the number of bytes transferred	I/O routine	I/O routine	DMA controller
6	Termination of data transfer	I/O routine software	I/O routine software	DMA controller
7	Indication of completion of data transfer to the software	Not applicable	Not applicable	Device controller; raises interrupt to the CPU at the end of data transfer

10.4.5 DMA Controller in Micros

In microcomputers, the DMA hardware is basically a special programmable controller, which performs mainly coordination of data transfer. Control functions like device selection are not done by the DMA controller. Hence, the software in a micro should perform such operations which are performed by the data channel in a large system. Fig. 10.44 presents the block diagram of a DMA controller organisation. The DMA controller's main functions are as follows:

1. Accepting the DMA parameters issued by the CPU (software), and storing them in its internal registers.
2. Performing the data transfer between the memory and the I/O controller, as and when the I/O controller raises a request (DMA request).
3. Once the number of bytes specified by the 'byte count' parameter has been transferred, informing the I/O controller the 'BYTE COUNT ZERO' information.

Issuing the DMA parameters to the DMA controller is achieved by the software by means of OUT instructions. The DMA controller has several registers to store the DMA parameters. Each register stores a specified parameter. Each of these registers is designed as an output port which is assigned a different port address. Hence, to issue a DMA parameter, the software issues an OUT instruction for the corresponding output port. In other words, to the software, the DMA controller appears to be a set of output ports, whose function is accepting and maintaining the DMA parameters. Thus, a DMA controller has two distinct roles:

1. Responding to DMA initialization sequence by the CPU software.
2. Co-ordinating the data transfer between the memory and the I/O controller.

To perform an I/O operation, software first initializes the DMA controller by means of a series of OUT instructions. Then, the software gives the command and the relevant command parameters to the device controller.

How does the DMA controller know which I/O controller should be involved? There are two schemes:

Scheme 1: Having a separate DMA controller, for both floppy disk controller and hard disk controller, as shown in Fig. 10.45. Each DMA controller takes care of supporting one of the I/O controllers. Each DMA controller has a set of registers for storing the DMA parameters. Thus, the software deals with two independent DMA controllers.

Scheme 2: Having a single DMA controller, with multiple sections or multiple channels, as shown in Fig. 10.46. In this scheme, physically, there is one DMA controller but logically there are two or more DMA controllers.

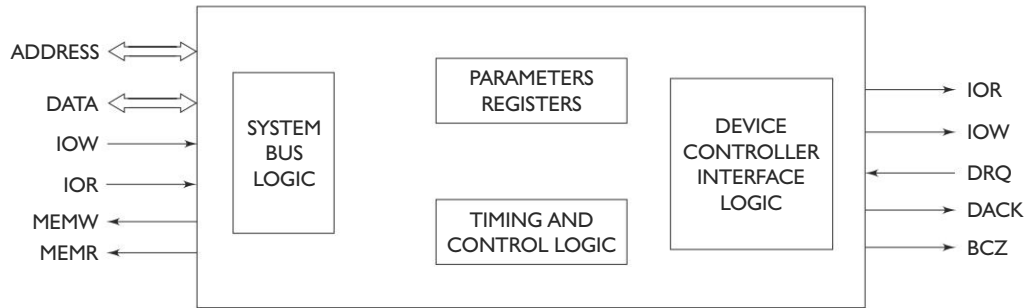


Fig. 10.44 DMA controller block diagram

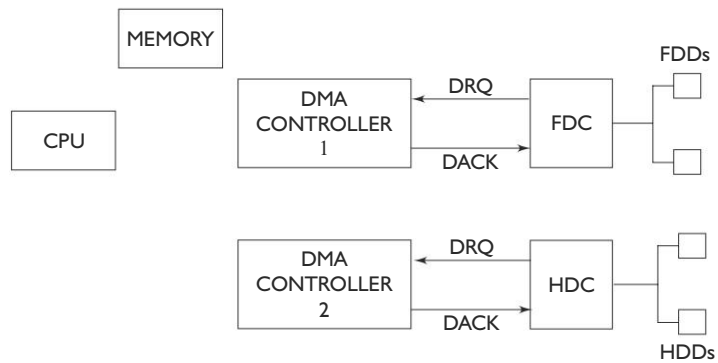


Fig. 10.45 Independent DMA controllers

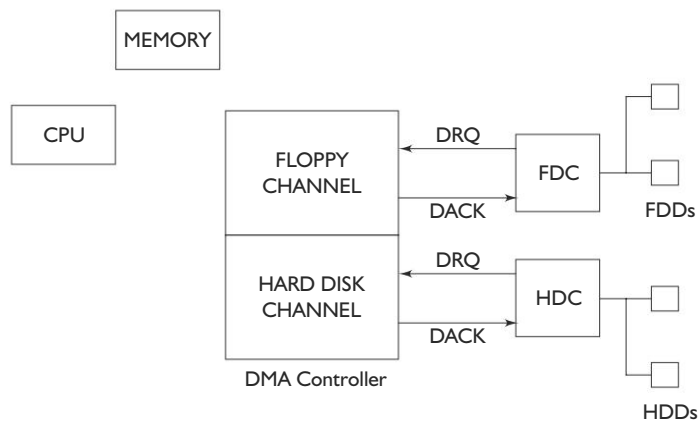


Fig. 10.46 Logical DMA channels in a DMA controller

The software does not distinguish between the two types of schemes, since logically there are two set of DMA channels. Since, in the second scheme, the two DMA channels are physically integrated into one DMA controller, extent of parallelism between the two DMA channels are curtailed. Thus, the different DMA channels inside a DMA controller can work concurrently but not simultaneously.

The IBM PC design adopts the second scheme. A single DMA controller chip, INTEL 8237A, with four DMA channels, is used. Figure 10.47 shows the configuration of the DMA controller in IBM PC. For each channel, it provides multiple modes of operation and the choice of the mode is programmable. Since the DMA controller interacts with the memory via the system bus in the PC, at a given instant, only one channel is serviced. However, all the channels can be initiated and can be active simultaneously. The 8237A maintains a priority scheme which decides priority in case of simultaneous requests for more than one channel.

To the system software, the 8237A appears as if there are four independent DMA channels. Each DMA channel is assigned a set of I/O ports for maintaining the DMA parameters. Once a DMA channel is initialised by the software, it is ready for performing data transfer between the associated device controller and the memory. However, until the device controller makes a request (DRQ), the DMA channel is in an idle state. On receiving the DRQ, the DMA controller services the channel subject to its priority. To perform data transfer between the memory and the device controller, the DMA controller needs to *take over* the system bus.

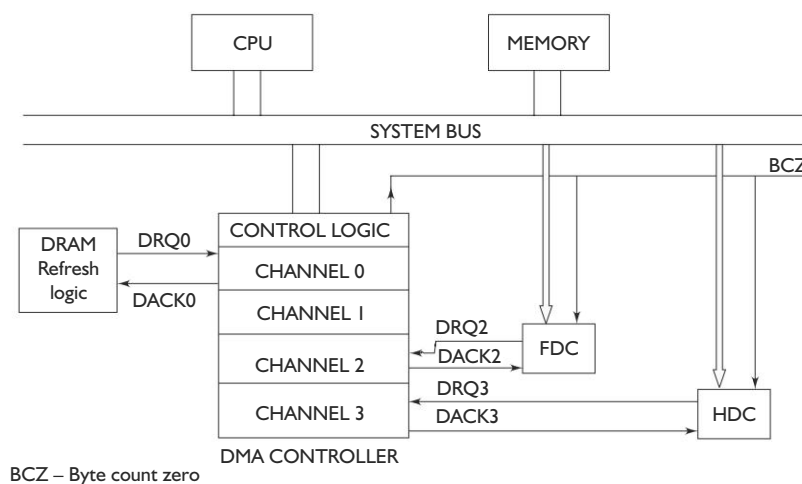


Fig. 10.47 Configuration of DMA controller in PC

10.4.5.1 Assignment of DMA Channels in PC

Figure 10.48 illustrates the function of the four DMA channels in the PC. The channel 0 is utilized for a non-data transfer function, namely refreshing dynamic memory in the system. The channel 1 is not used in the IBM PC. Certain PCs use it for supporting data transfer with special peripherals such as scanner. The channel 2 is used for data transfer between floppy disk controller and memory. The channel 3 is used for data transfer between hard disk controller and memory.

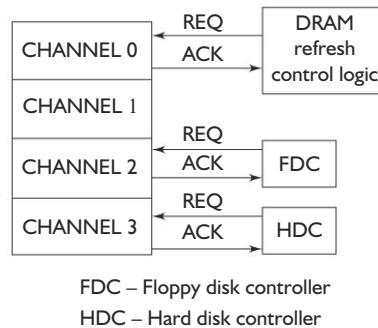


Fig. 10.48 Use of DMA channel in PC

As such the DMA controller is not aware of the exact function of each channel. It assumes that a device controller is linked to each channel. All channels are treated equally except for the priority. The channel 0 is given highest priority followed by channel 1, 2 and 3. The channel 3 has the least priority.

10.4.5.2 Operating Modes of DMA Channels

All the four DMA channels are identical and their capabilities are same. The only difference is the order of priority of service when multiple DMA channels simultaneously need the bus for DMA cycle. Each channel can operate in three different modes:

1. Single byte mode or multiplex mode
2. Block mode or burst mode
3. Demand mode

Single Byte Mode

In this mode, the DMA channel uses the bus for one byte of data transfer. Then, the bus is surrendered back. When next byte is to be transferred, the bus has to be obtained again.

Block Mode

In this mode, the DMA channel keeps the bus control continuously till all the bytes are transferred.

Demand Mode

This mode results in optimum use of the bus. Once the bus control is obtained for transferring one byte of data, the return of the bus after completion of data transfer depends on the actual need. If the device controller is ready for next byte of data transfer, the bus is not returned. Otherwise, the bus is retained. In other words, the bus is not unnecessarily retained unlike the burst mode.

Figure 10.49 illustrates the action taken by DMA channel in different modes.

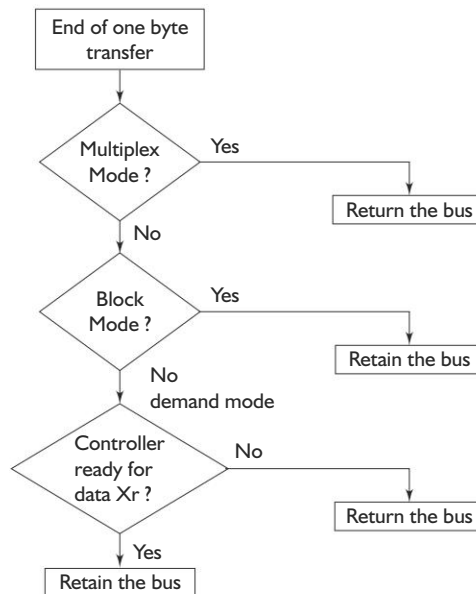


Fig. 10.49 DMA channel modes



10.5 Bus Arbitration Techniques

Bus arbitration is the process of determining the bus master who gets the bus control, at a given time, when there is a request for bus mastership from one or more bus masters. A bus master can initiate and perform a bus cycle for data transfer. A slave cannot become a bus master and hence cannot initiate a bus cycle. In most computers, the processor and DMA

controller are bus masters, whereas memory and I/O controllers are slaves. In some systems, certain intelligent I/O controllers are also bus masters. In some systems, multiple processors share a bus. Each processor can become a bus master.

When more than one bus master simultaneously needs the bus, only one of them gains control of the bus and become active bus master. The others should wait for their turn. The 'bus arbiter' decides who would become current bus master. There are two types of bus arbiters:

1. *Centralized bus arbitration* in which a dedicated arbiter has the role of bus arbitration.
2. *Distributed bus arbitration* in which all bus masters cooperate and jointly perform the arbitration. In this case, every bus master has an arbitration section.

10.5.1 Methods of Bus Arbitration

There are three popular schemes for bus arbitration used in computer systems:

1. Daisy Chain Method
2. Polling or Rotating Priority Method
3. Fixed Priority or Independent Request Method

The arbitration response time and hardware cost differ in different methods.

10.5.1.1 Daisy Chain Method

The 'Bus Request' (BR) signals of all the masters form a one line bus, BR. Similarly the 'Bus Busy' (BB) output of all masters form another one line bus, BB. When no bus master is using the bus, the BB is inactive. Whenever one or more bus request is present, the BR becomes active. The arbiter initiates the arbitration by transmitting the BG (bus grant) signal. It reaches the first master. Now there are two possibilities:

1. The first master does not need the bus and has not raised BR. It simply passes the BG signal to next bus master.
2. The first master needs to use the bus now and has raised BR. It becomes current bus master by issuing the BB signal, and dropping BR signal. However, the BR may be still active if some other master also has raised BR. The current bus master retains the bus till it completes its bus operations. On seeing the BB active, all other bus masters understand that some bus master has taken over the bus, and hence they keep waiting. As soon as the current bus master completes its use of the bus, it removes the BB signal. If the BR is still active, again the arbiter sends out the BG signal.

Fig. 10.50 illustrates the daisy chain method.

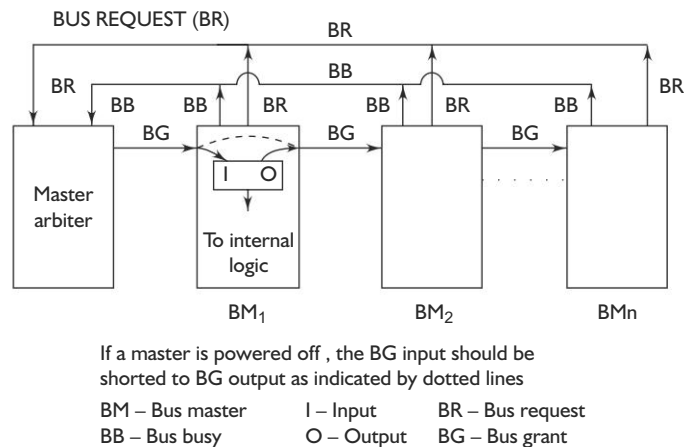


Fig. 10.50 Daisy chained bus arbitration

The daisy chain method is a low cost technique. Only the BG line is daisy chained; the BR and BB are shared bus lines. Any number of masters can be handled. The priority for a master depends on its proximity to the arbiter (BG signal). The blemish of this technique is the nuisance that is created by the bus master close to arbiter by frequent requests. Due to this, the bus is always given to such a master, that affects the service to other masters. Hence care should be taken in the wiring (routing) order for the BG line. Generally, the masters which need the bus less frequently are given priority and located close to the arbiter. Any master that is powered-off simply passes the BG signal to the next master.

10.5.1.2 Polling Method

In the polling method (Fig. 10.51), the BR and BB signals are similar to those in daisy chain method. Once the BR is active, the arbiter polls the masters, one-by-one, transmitting the address of the master, on address lines. The master, whose address matches with the address issued by the arbiter, becomes the current bus master by generating the BB signal. This method needs more lines since address has to be transmitted. But the order of addressing the masters (polling sequence) is programmable.

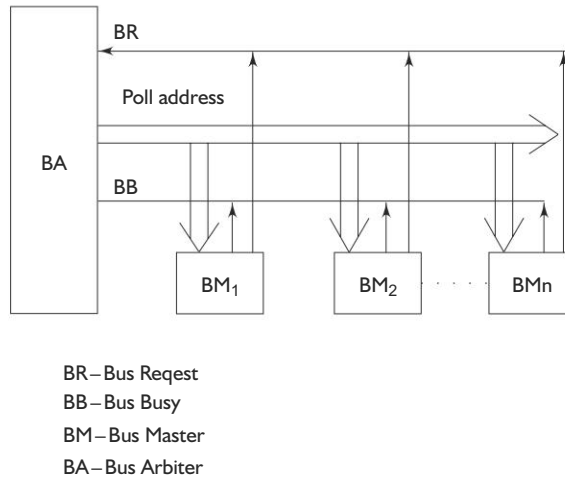


Fig. 10.51 Bus arbitration-polling method

10.5.1.3 Independent Request Arbitration

Fig. 10.52 shows the independent request method. Each master has a dedicated BR output line and BG input line. If there are n masters, the arbiter has n BR input and n BG output. The arbiter follows a priority order, with different priority level to each master. At a given time, the arbiter issues bus grant to the master of highest priority among the masters who have raised bus requests. This scheme needs more hardware but generates fast response. The priority order is also programmable.

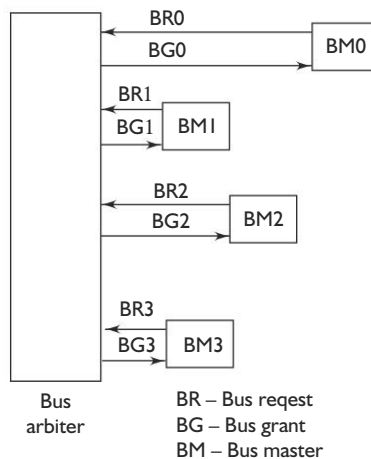


Fig. 10.52 Arbitration by fixed priority method

10.6 Bus Concepts

A bus is a communication mechanism that enables multiple functional units in a computer to send data and other signals to each other. It is a modern concept. The purpose and nature of communication between different hardware units in a computer is listed in Table 10.7. Table 10.8 compares a bus structure and non-bus structure. The bus concept is popular with minicomputers and microcomputers due to reduction of hardware cost.

TABLE 10.7 Communication in a computer

Sl. no.	Communicating units	Type of communication/Purpose
1	CPU and memory	(a) Instruction fetch (b) Operand fetch (c) Result storing
2	CPU/memory and input unit	(a) Program or data input (b) Issuing commands to device/controller (c) Reading device/controller status
3	CPU/memory and output unit	(a) Program or data output (b) Issuing commands to device/controller (c) Reading device/controller status

TABLE 10.8 Bus structure and non-bus structure

Sl. no.	Bus structure	Non-bus structure
1	Many units share a common set of lines; hence cheaper	Wires are more; hence costlier
2	An additional control circuit is needed to avoid clash, for example, when CPU is reading an instruction from memory, data should enter only CPU and not other units	No additional logic is necessary since clash is not possible
3	Simultaneous communication is not possible; hence transfer speed is less	Simultaneous communication is possible; hence faster

In large systems, multiple isolated paths of communication exist between any two units in the hardware. Hence, several units can communicate simultaneously, which offer high performance. In most minicomputers, there are two types of buses:

- (a) memory bus
- (b) I/O bus

The CPU and memory communicate among themselves over the memory bus which is designed for a high speed of operation. The I/O bus is designed for a lesser bandwidth since

it handles relatively slower I/O transfers. This is illustrated in Fig. 10.53. In early microcomputers, universal bus concept is used as shown in Fig. 10.54.

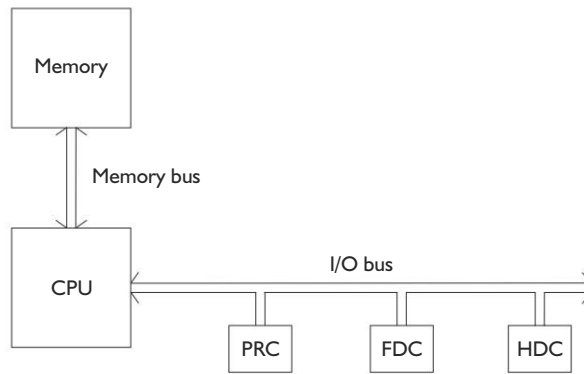


Fig. 10.53 Bus concept in mini computers

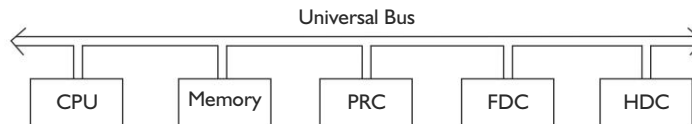


Fig. 10.54 Bus concept in microcomputers

10.6.1 Bus Cycle

The bus cycle is a sequence of events, on the bus, for transferring one word of information between the CPU and the memory (or I/O port). The sequence is monitored by the CPU. It starts the bus cycle, goes through the defined steps and ends the bus cycle when the communication is over. The different types of bus cycle are

- Memory Read
- Memory Write
- I/O Read
- I/O Write
- Interrupt Acknowledgment

The *Memory Read* bus cycle is performed for reading one byte (or word) from memory.

The *Memory Write* bus cycle is performed for writing one byte (or word) into the memory.

The *I/O Read* bus cycle achieves reading one byte of data from an input port.

The *I/O Write* bus cycle performs writing one byte of data into an output port.

The interrupt acknowledgment bus cycle transfers vector code.

Table 10.9 lists types of bus cycles involved in executing different instructions.

TABLE 10.9 Types of bus cycles when executing an instruction

Sl. no.	Instruction Type	Phase	Bus cycle/action
1	ADD instruction	(a) Fetch (b) Decode (c) Fetch operand (d) ADD (e) Store result	Memory read bus cycle Internal operation Memory read bus cycle Internal operation Memory write bus cycle
2	IN instruction	(a) Fetch (b) Decode (c) Execution	Memory read bus cycle Internal operation I/O read bus cycle
3	JUMP instruction	(a) Fetch (b) Decode (c) Execution	Memory read bus cycle Internal operation Internal operation
4	OUT instruction	(a) Fetch (b) Decode (c) Execution	Memory read bus cycle Internal operation I/O write bus cycle

10.6.2 Bus Organization

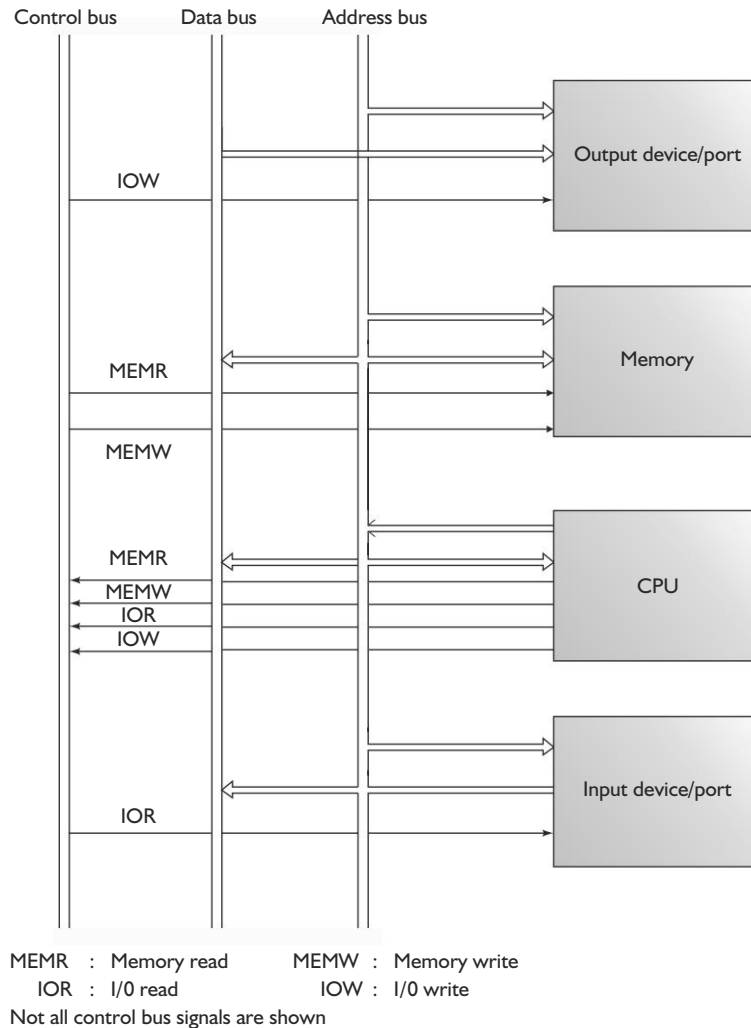
A bus is a group of wires/lines/tracks carrying signals common to several hardware units. The signals are divided into three different types as illustrated in Fig. 10.55:

- (a) Address bus signals
- (b) Data bus signals
- (c) Control bus signals

Address Bus

The address bus is used for two different purposes as follows:

- (a) To carry memory address, while reading from or writing into memory.
- (b) To carry I/O port address or device address, while sending/reading data to/from I/O port.

**Fig. 10.55** Bus concept

In early microprocessors, the address bus is uni-directional since only the CPU can send address and other units cannot address the microprocessor. The advanced microprocessors that have on-chip cache memory have bi-directional address bus, for cache memory invalidation. In addition to processor, the DMA controller can send memory address, during its bus cycle, for data transfer between memory and an I/O controller. The DMA controller is not shown in Fig. 10.55, for simplicity.

Data Bus

The data bus is a bi-directional bus and is used for the following tasks:

- (a) To fetch instruction from memory.
- (b) To fetch operand (of an instruction) from memory.
- (c) To store the result (of an instruction) into memory.
- (d) To send a command to an I/O device controller (or port).
- (e) To read the status from an I/O device controller (or port).
- (f) To read data from a device controller (or port).
- (g) To issue data to a device controller (or port).

During a bus cycle, if the CPU is the bus master, the CPU either places data on data bus or takes data from data bus. Though the DMA controller also can become a bus master, it can neither supply data to data bus nor receive data from data bus. It can only direct the memory and an I/O controller to supply/receive data. However, during CPU's bus cycle, the DMA controller can interact with data bus, if it is addressed by the CPU.

Control Bus

The different control signals in a bus are shown in Fig. 10.56 and are briefly explained below:

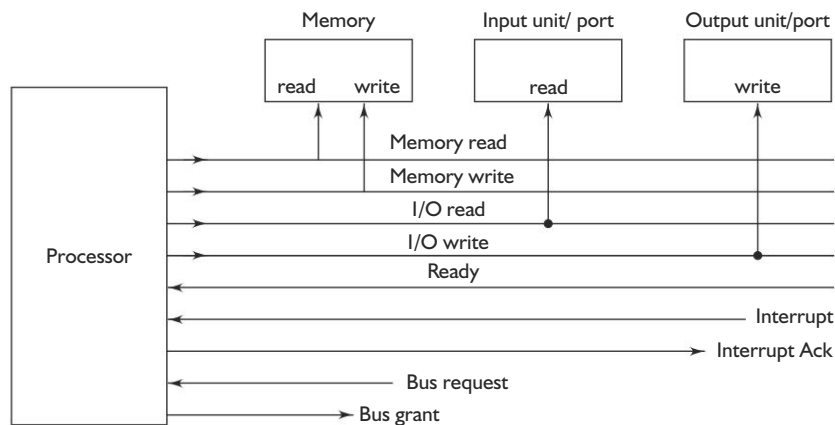


Fig. 10.56 Control bus signals

(a) Memory Read

This signal is issued by the CPU (or DMA controller) when performing a read operation with the memory. It is an input signal to the memory. The memory reads out the data, from the location whose address is present on the address bus.

(b) Memory Write

This signal is issued by the CPU (or DMA controller) when performing a write operation with the memory. It is an input signal to the memory. The memory stores the data in the location whose address is present on the address bus.

(c) I/O Read

This signal is issued by the CPU when it is reading from an input port. The input port addressed by the CPU places its data on the data bus. The DMA controller uses I/O read signal to command an I/O controller to place data on data bus during a DMA cycle for data transfer.

(d) I/O Write

This signal is issued by the CPU when writing into an output port. The output port addressed by the CPU receives the data from the data bus. The DMA controller uses I/O write signal to command an I/O controller to accept data from data bus during a DMA cycle for data transfer.

(e) Ready

The ready is an input signal to the CPU, generated in order to synchronize the slow memory or I/O ports with the fast CPU. For instance, when the CPU is reading from a relatively slow memory, the READY signal is used to “freeze” the CPU till the memory completes the read operation as shown in Fig. 10.57. In other words, delay is introduced in the bus cycle (in memory read sequence) by indicating that the memory data is not yet ready.

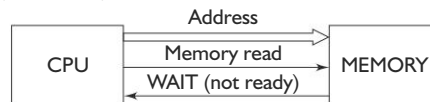


Fig. 10.57 Synchronization between a fast CPU and a slow memory

(f) Other Signals

Various other signals such as Interrupt Requests and DMA Requests are also considered as control signals. These are discussed in other relevant sections.

10.6.3 Bus Cycle Timing

Under normal conditions, the CPU completes the bus cycle in a fixed time interval. The sequence of operations performed over a bus cycle are divided into many stages, known as

bus states. In the case of INTEL 8088, there are four states, in a bus cycle, named as T1, T2, T3 and T4. The time duration of each state is equal, and it is equal to the period of the clock signal. The microprocessor in the original IBM PC operates at 4.77 MHz clock frequency and the duration of each state is 210 ns.

During T1, the CPU starts the bus cycle by sending the memory address (or I/O port address) on the address bus. In T2, the CPU generates one of the four control signals: Memory Read, Memory Write, I/O Read, or I/O Write. If the bus cycle is of memory write or I/O write type, the CPU releases the data on the data bus, in T2. In T3, the addressed memory or I/O port reads from the data bus. In memory read or I/O read bus cycle, the addressed memory (or I/O port) presents the data on the data bus in T3 and the CPU reads this data from the data bus, in T3. In short, the addressed port or memory does data transfer operation (read or write) with the bus in T3. In T4, the CPU winds up the bus cycle by removing all the signals issued by it earlier. Fig. 10.58 shows the activities in a bus cycle.

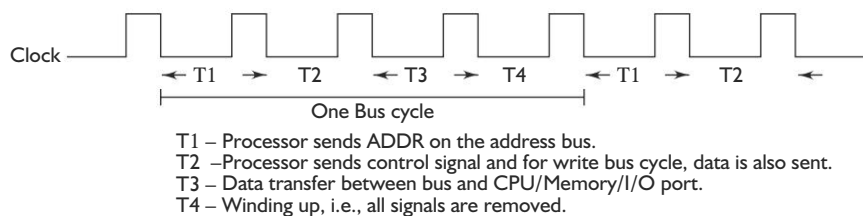


Fig. 10.58 Bus cycle actions

The memory (I/O port) should be fast enough to respond with data in T3 during a read operation. In other words, the access time of memory should be less than the interval from the beginning of T2 to the beginning of T3.

Suppose the memory access time is more than this interval. In such a case, the CPU has to be informed not to read data in T3. This is done by alerting the CPU in T2, that the addressed memory will not be ready for data transfer in the next state, i.e. T3. In the beginning of T3, the processor senses the status of READY input. If READY is low, the processor understands that the memory needs more time to complete the read operation. Hence, in T3, the processor does not read from the data bus. Instead, the processor introduces an additional state between T3 and T4. The additional state is known as wait state, Tw. In the beginning of Tw, the processor again checks the READY signal. If READY signal is high, the processor performs the data transfer in Tw. Otherwise, the CPU introduces an additional wait state.

Thus, until the READY input remains low, the CPU continues to be in wait state. Once the READY is HIGH, then the CPU reads the data bus contents and proceeds to T4, which is the winding up state. The duration for which the CPU is placed in wait state depends on the access time of the memory. The wait state duration is expressed as the number of wait states, which is equal to the number of clock cycles elapsed between T3 and T4. Fig. 10.59 shows a bus cycle with two wait states.

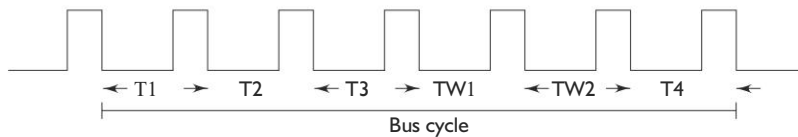


Fig. 10.59 Bus cycle with two wait states

10.6.3.1 Zero Wait State

The zero wait state implies that the CPU does not enter into wait state during its communication with memory and I/O ports.

In some systems, the memory may be fast with “zero wait state” but certain I/O controllers/ports may need “one or more wait state”.

Example 10.1 A microcomputer based on Intel 8088 microprocessor operates at zero wait state at a clock frequency of 8 MHz. Calculate the maximum bandwidth offered by the system.

Clock frequency = 8 MHz.

Clock period = $1/(8 \times 10^6) = 125 \text{ ns}$.

Each bus cycle transfers one byte, and there are four states (clock periods) in one bus cycle, since we have a zero wait state system.

Hence, duration of a bus cycle = $4 \times 125 \text{ ns} = 500 \text{ ns}$.

This is the time for transferring one byte.

Hence, bandwidth = number of bytes transferred in one second

$$= 1/(500 \text{ ns}) = 2 \times 10^6$$

$$= 2 \text{ MB/s.}$$

10.6.4 Asynchronous and Synchronous Transfer

When two units communicate with each other for data transfer, usually one of them is the master, and the other one, slave. One word of data transfer is done in one clock period or sometimes in more than one clock period. There are two types of data transfer depending on the mechanism of timing the data: Synchronous and Asynchronous. The synchronous transfer is possible between two units each of which knows the behavior of the other. The asynchronous transfer enables communication between two unknown types of units.

10.6.4.1 Synchronous Transfer

In *Synchronous method* of transfer, the sending and receiving units are supplied with same clock signal. The master performs a sequence of actions, for data transfer, in a predetermined order; each action is synchronized to either the raising edge (or falling edge) of the clock. The master is designed to supply the data at a time when the slave is definitely ready for it. If necessary, the master will introduce sufficient delay to take into account the slow response of a slave, without any request from slave. When the master sends data to the slave, the data is transmitted by the master, without expecting any acknowledgment from the slave. Similarly, when the master reads data from the slave, neither the slave informs that it has placed data nor the master acknowledges that it has read the data. Both master and slave perform 'placing' and 'taking' data at defined edges of the clock signal. Since both master and slave know the response time of each other, no confusion is possible. However, before initiating the data transfer, the master should logically select the slave either by sending slave's 'address' (in the case of bus type communication) or sending 'device select' signal to the slave. The device is selected but there is no acknowledgement from slave to master.

Fig. 10.60 shows timing diagram for synchronous transfer read operation. The master transmits slave address and read signal at the raising edge of a clock. The slave places data at the falling edge of the clock. The master removes the address and read signal in the raising edge of the next clock and the slave also removes its data on this edge. The entire read operation is over in one clock period. Fig. 10.61 shows timing diagram for synchronous write operation. The master places following three different items on the raising edge of the clock:

1. Slave address
2. Data
3. Write signal

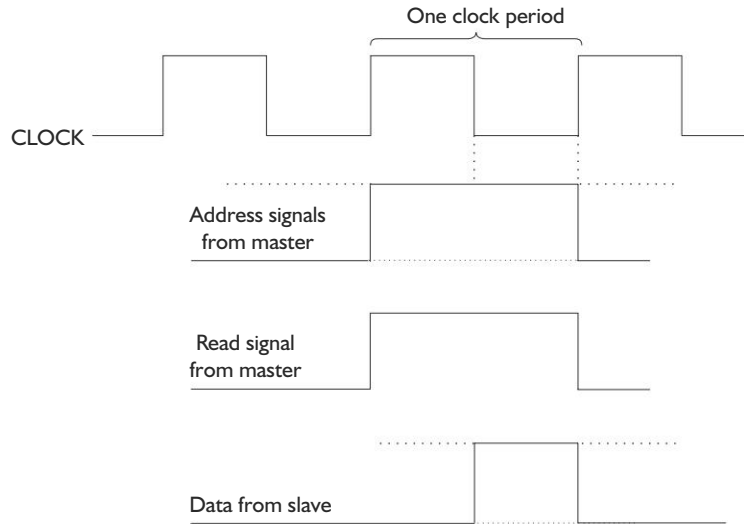


Fig. 10.60 Synchronous transfer timing diagram for read operation

The slave reads the data before the end of the clock cycle, since the master removes all its signals in the next clock. If the slave cannot respond within a clock due its high response time, the master allows the data transfer cycle to spread over multiple clock periods. Fig. 10.62 shows a multiple cycle synchronous read operation.

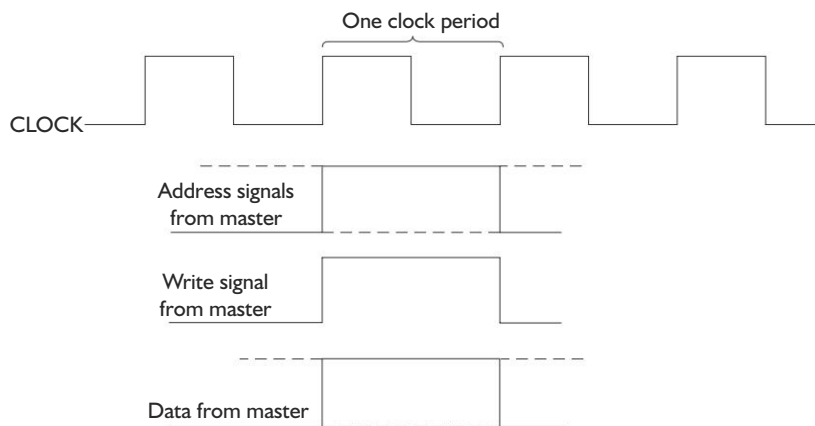
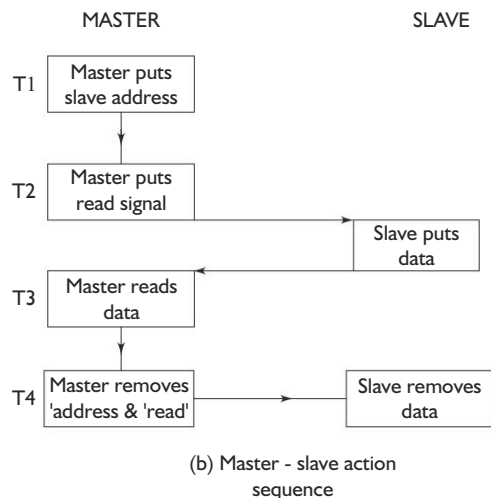
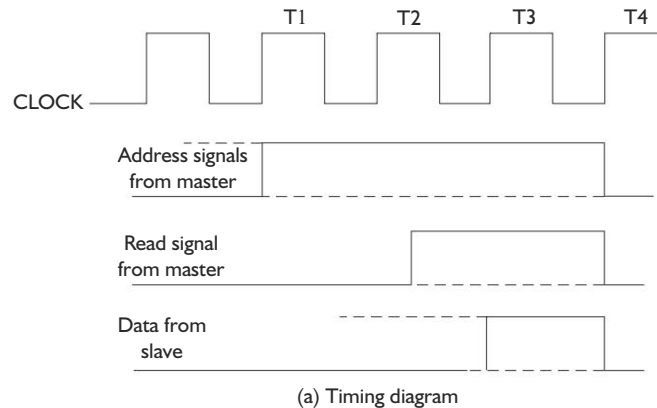


Fig. 10.61 Synchronous transfer timing diagram for write operation

**Fig. 10.62** Synchronous read in multiple clocks

The following are some examples of synchronous transfers (Fig. 10.63):

1. A processor writing to an output port.
2. A processor reading from an input port.
3. A processor reading from main memory.
4. *A processor writing into main memory for example, 8088 Bus cycle:* The 8088 performs a bus cycle as a series of bus states: T1, T2 etc. The duration of each state is equal to one clock period. The 8088 gives address in T1 and control signal and data in T2. It expects memory to accept the data in T3 and hence winds up the bus cycle in T4.
5. *Data transfer between floppy controller and floppy drive:* The floppy controller transmits serial data (to be written) to the floppy disk drive on the MFM WRITE DATA line

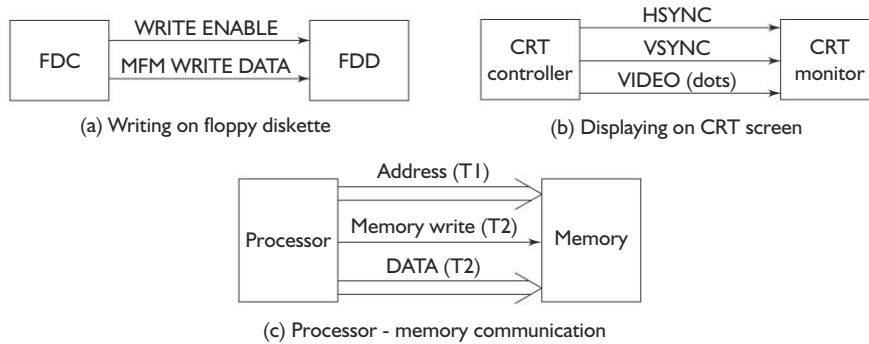


Fig. 10.63 Synchronous transfer examples

at a rate that matches the rotational speed of the floppy disk drive. The frequency of write clock in floppy controller is chosen accordingly. Before starting the data stream, the floppy controller makes WRITE ENABLE (WRITE GATE) signal active. On sensing this signal, the floppy drive starts receiving the data bits. There is no acknowledgement from floppy disk drive about this. Both floppy controller and floppy disk drive operate with same clock frequency. They generally have separate clock sources but both the clocks are synchronized by means of special sync bit pattern written on the floppy media.

6. *Displaying data on CRT screen:* The CRT controller sends the display dot information on VIDEO line as continuous 1's and 0's. The dot clock in the CRT controller is chosen to match the bandwidth of the CRT monitor. The HSYNC and VSYNC signals are not for synchronizing the data transfer. They synchronize the electron beam in the CRT monitor with the clock signal in the CRT controller so that the data display on the screen is aligned properly.

Merits of synchronous transfer are as follows:

1. The design is straight forward. The master need not look for any signal from the slave, though the master waits (add delay) for a time equal to slave's response time.
2. The slave need not generate acknowledgment, though it has to follow the timing as per the rules fixed by the master.

Demerits of synchronous transfer are as follows:

1. If the slave operates at a lower speed, the master waits for a long time during the transfer sequence.
2. If one unit connected to a bus is a slow speed unit, the rate of transfer to all other units is limited, by the slow speed unit. The bus clock frequency takes into account the slowest unit on the bus.

10.6.4.2 Asynchronous Transfer and Handshaking

In asynchronous transfer, there is no common clock between the master and slave. They follow some sort of acknowledgment protocol during data transfer sequence. When the master sends data, it informs the slave by means of a ACCEPT STROBE signal indicating that it has placed data on data lines. In response, the slave receives the data from the data lines. Figure 10.64 shows a source unit initiated asynchronous transfer. The source takes care of proper timing delay between the actual data signals and the ACCEPT STROBE. It places the data first, and after some delay, generates the STROBE. Similarly, before

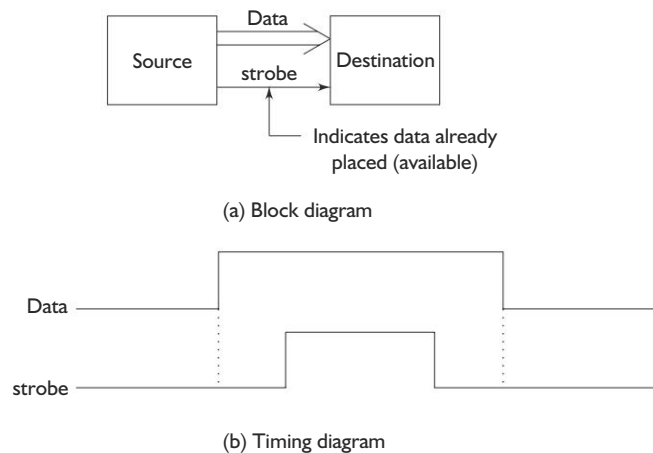


Fig. 10.64 Source initiated asynchronous data transfer

removing the data it removes the strobe and after some delay it removes the data. These two (leading and trailing end) delays ensure reliable data transfer between generating and receiving ends. A destination unit also can initiate data transfer by sending a REQUEST STROBE to the sending unit as shown in Fig. 10.65. In response, the source unit places data on data lines. After receiving data, the destination end removes the REQUEST STROBE. Only after sensing the removal of REQUEST STROBE, the source removes the data. This protocol is followed for every piece (byte or word) of data. There is a risk in these two methods. Consider the source initiated asynchronous transfer. If the source/destination has a fault, then one of the following problems may occur:

1. The destination does not sense the STROBE signal from the source. Hence, it does not take the data transmitted by the source. This is unknown to the source.
2. The source spuriously generates the STROBE, when there is no need to transfer data. Now, the destination simply reads the contents of data lines as *genuine* data without knowing about the problem.

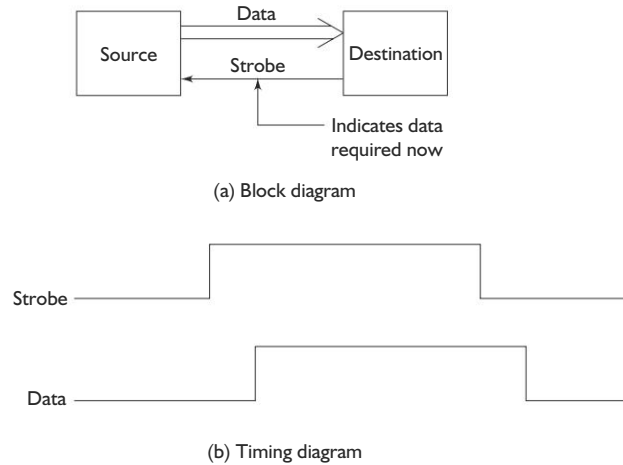


Fig. 10.65 Destination initiated asynchronous data transfer

Similar problems can occur in destination initiated transfer also. To overcome this, a handshake protocol is commonly followed between the source and destination. In handshaking protocol, both source and destination generate strobe and acknowledgment signals. Fig. 10.66 shows source initiated handshaking protocol. The source places data first and after some delay, issues DATA AVAIL signal. On sensing DATA AVAIL signal, the destination receives data and then issues DATA ACK signal indicating its acceptance of data. On sensing the DATA ACK signal, the source removes data and DATA AVAIL signal.

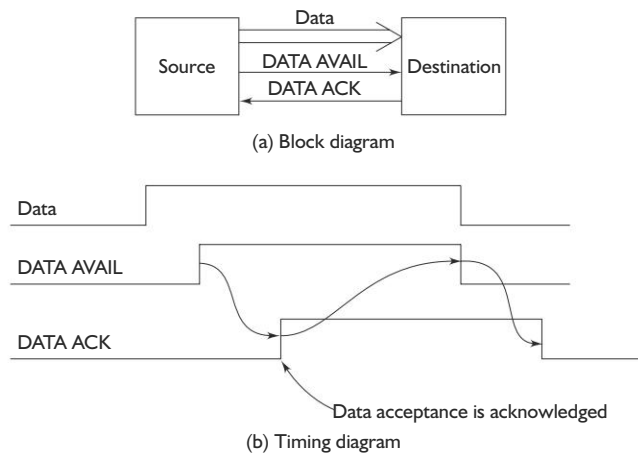


Fig. 10.66 Source initiated handshaking protocol

On sensing removal of DATA AVAIL, the destination removes the DATA ACK signal. Since both source and destination respond to raising or dropping of other end's signal, an abnormal action by one end is not a genuine sequence. Fig. 10.67 shows destination initiated handshaking protocol. The destination first sends the DATA REQ signal. On sensing this signal, the source places data and also issues the DATA AVAIL signal. On sensing DATA AVAIL signal, the destination acquires data and then removes the DATA REQ signal. On sensing this, the source removes both the data and DATA AVAIL signal.

Advantages of asynchronous transfer are as follows:

1. Since the source and destination follow handshaking, reliability of the transfer is assured.
2. A slow destination unit on a bus, does not slow down data transfer for a fast destination unit.

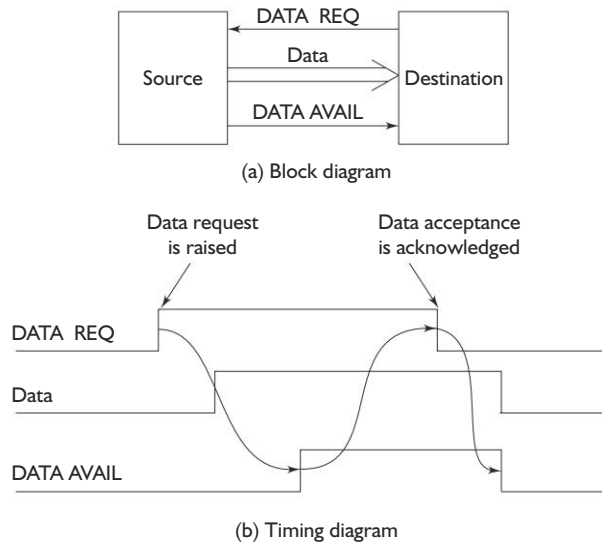


Fig. 10.67 Destination initiated handshaking protocol

Disadvantages of asynchronous transfer are as follows:

1. The design is more involved and costlier due to additional signals and sequence.
2. A slow destination unit on a bus can hold up the bus, whenever it gets a chance to communicate.

Examples of asynchronous transfer:

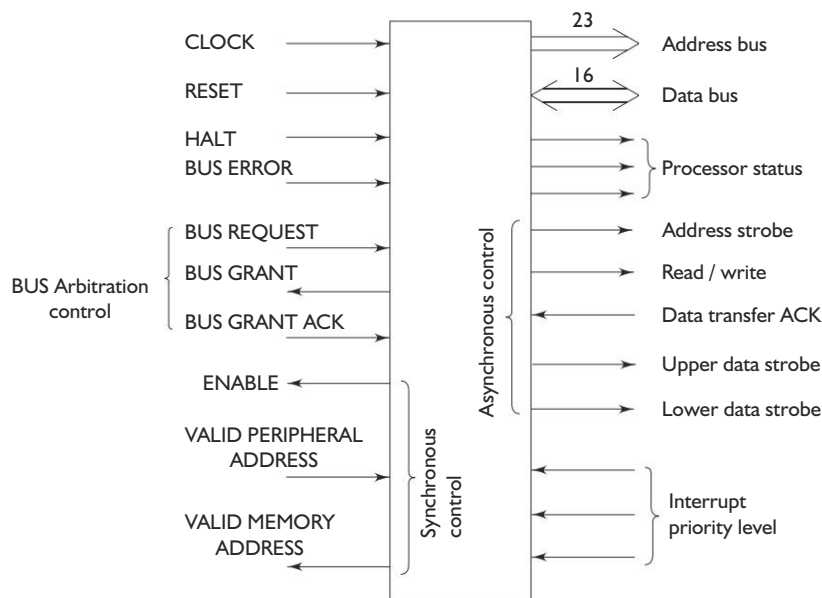
1. The centronics interface follows handshaking protocol; it is a popular parallel interface.
2. The bus transfers by most microprocessors such as Motorola 68010 and INTEL 80286.

10.6.5 CASE STUDY 2: Motorola 68010 and Bus Cycle

The Motorola 68010 is one of the members of the popular Motorola 68000 series. It has 16 bit data bus and 23 bit address. Fig. 10.68 shows the pinout of 68010. It follows asynchronous transfers as shown in Fig. 10.69. The 68010 also allows interfacing of 6800 peripherals as shown in Fig. 10.70, for synchronous transfer. This enables using 8 bit peripherals in a 68010 system.

10.6.5.1 68010 and Bus Arbitration

Fig. 10.71 illustrates bus arbitration sequence followed in a 68010 system.



The synchronous control pins enable interfacing of synchronous peripherals of 6800 with asynchronous 68010.

Fig. 10.68 Motorola 68010 bus signals

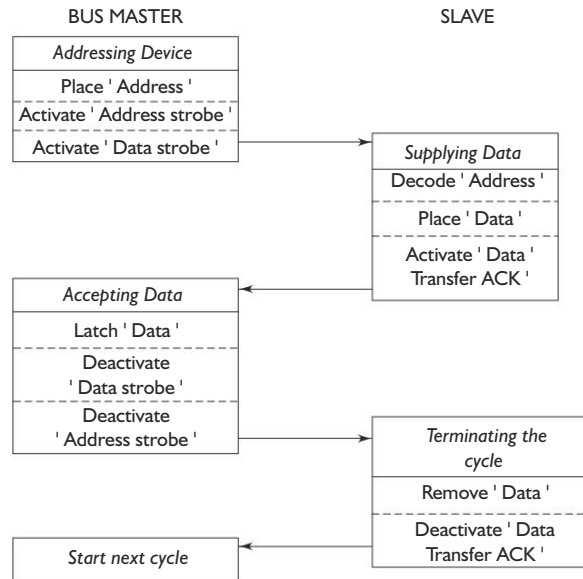


Fig. 10.69 Asynchronous read cycle by 68010

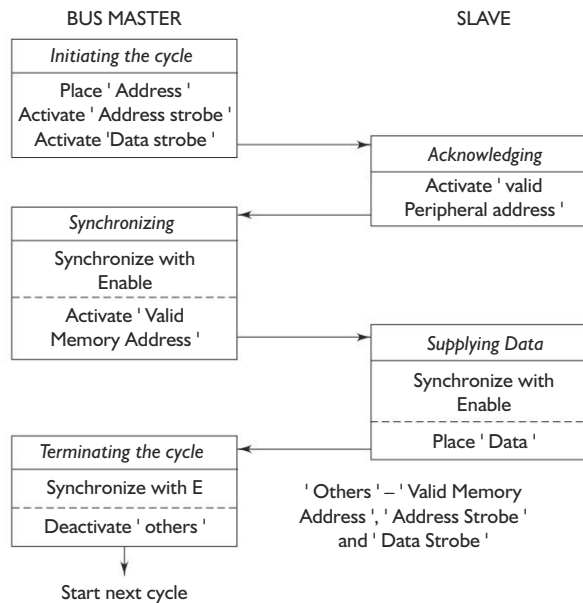


Fig. 10.70 Synchronous read cycle for 6800

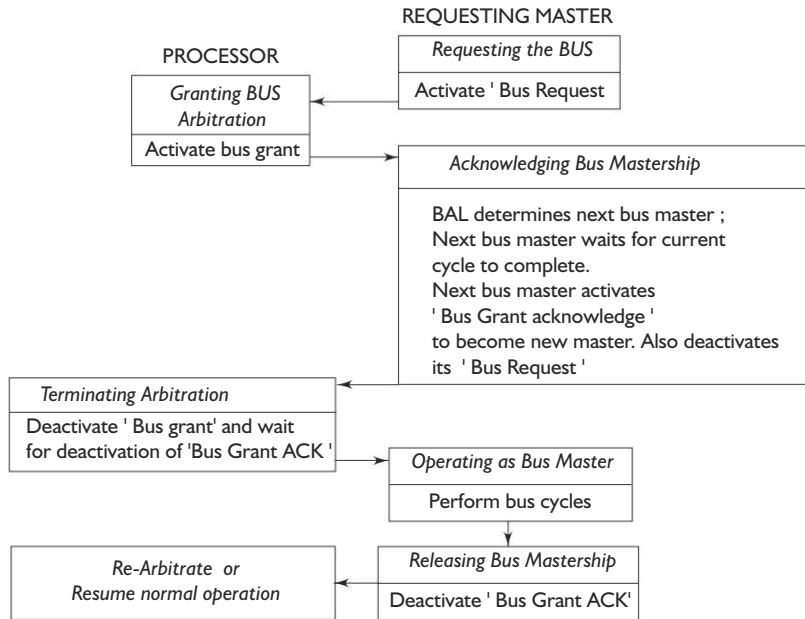


Fig. 10.71 Bus arbitration sequence for 68010 system

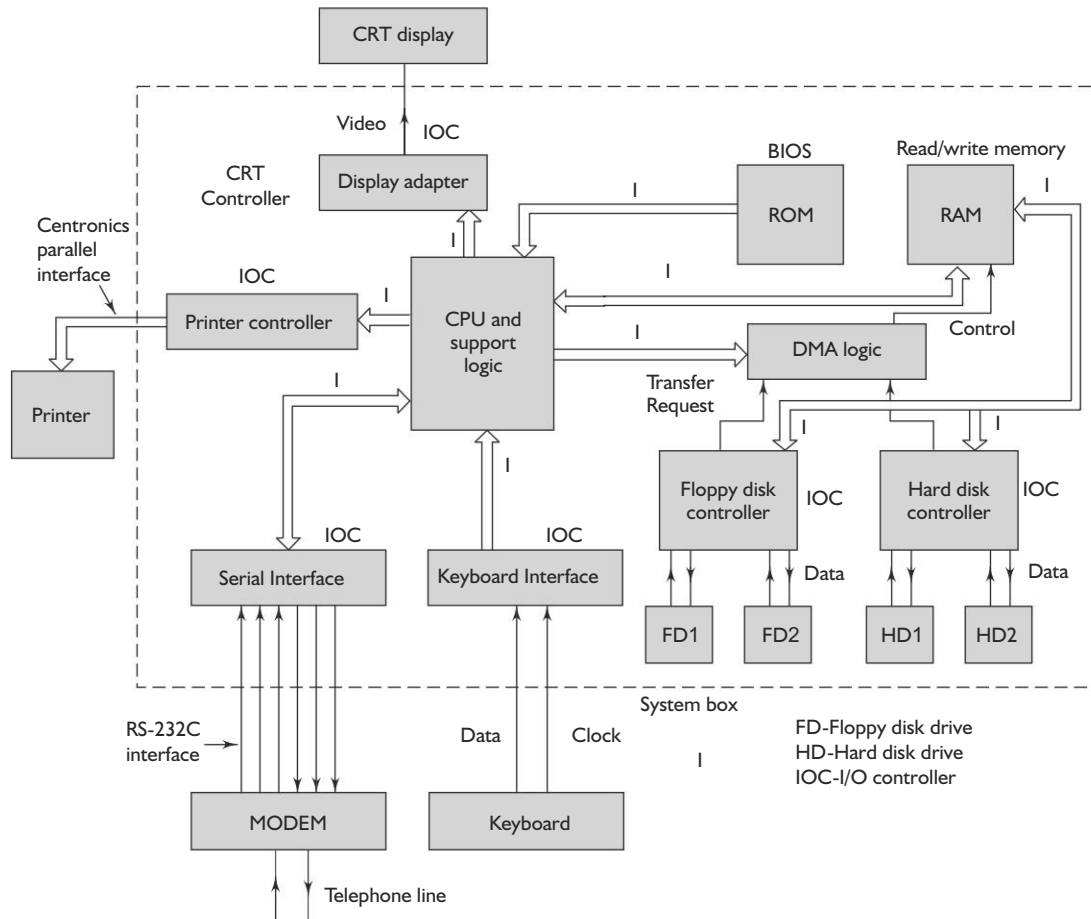
10.6.6 Multiple Bus Computers

The communication inside a modern computer system occurs over different types of buses listed below:

1. Local bus
2. System bus
3. I/O bus
4. Mezzanine bus

10.6.6.1 Early Style—Single Bus concept

Fig. 10.72 shows a functional block diagram of the PC and Fig. 10.73 illustrates the bus organization for the PC. Fig. 10.74 exhibits the signals on the PC bus. The original IBM PC operated at 4.77 MHz clock, with zero wait state for motherboard memory and one wait state for motherboard I/O ports. The access time of RAM is 150 ns.



Note: I denotes information that could be instruction, data, command, status or address. This figure mainly focusses on data. Not all I/o controllers are shown.

Fig. 10.72 Functional block diagram of a PC

The number of wait states required depends on the RAM access time and clock speed. The wait state requirement for any daughterboard memory or daughterboard I/O port is taken care by that daughterboard design. It can generate 'wait state request' signal.

In the 8088-PC, the system bus and I/O bus are the same. Fig. 10.75 shows the bus organization for the original PC motherboard. The floating point coprocessor is connected to the local bus of 8088. All other subsystems are linked to the processor through buffers. The

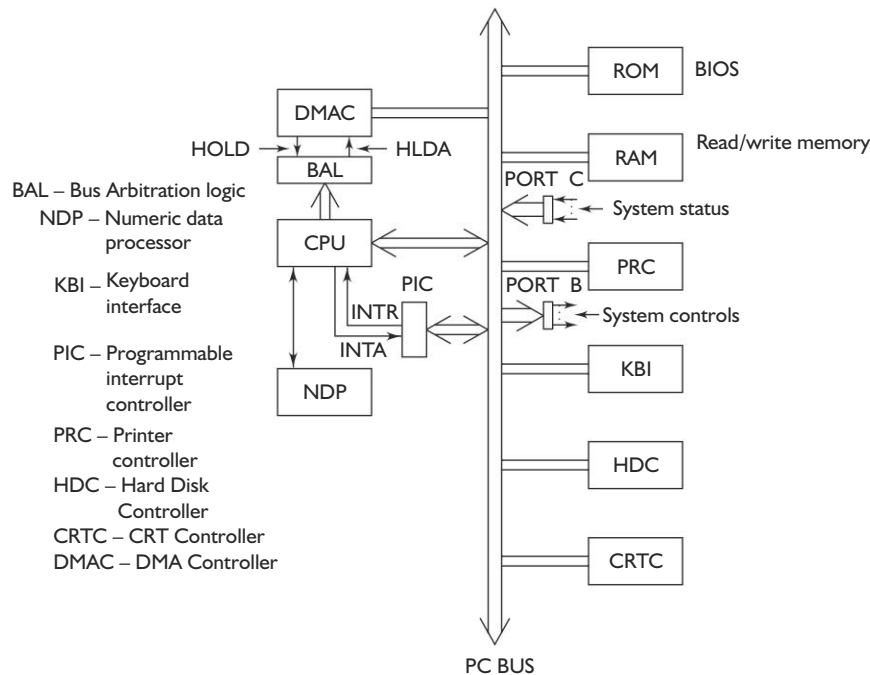


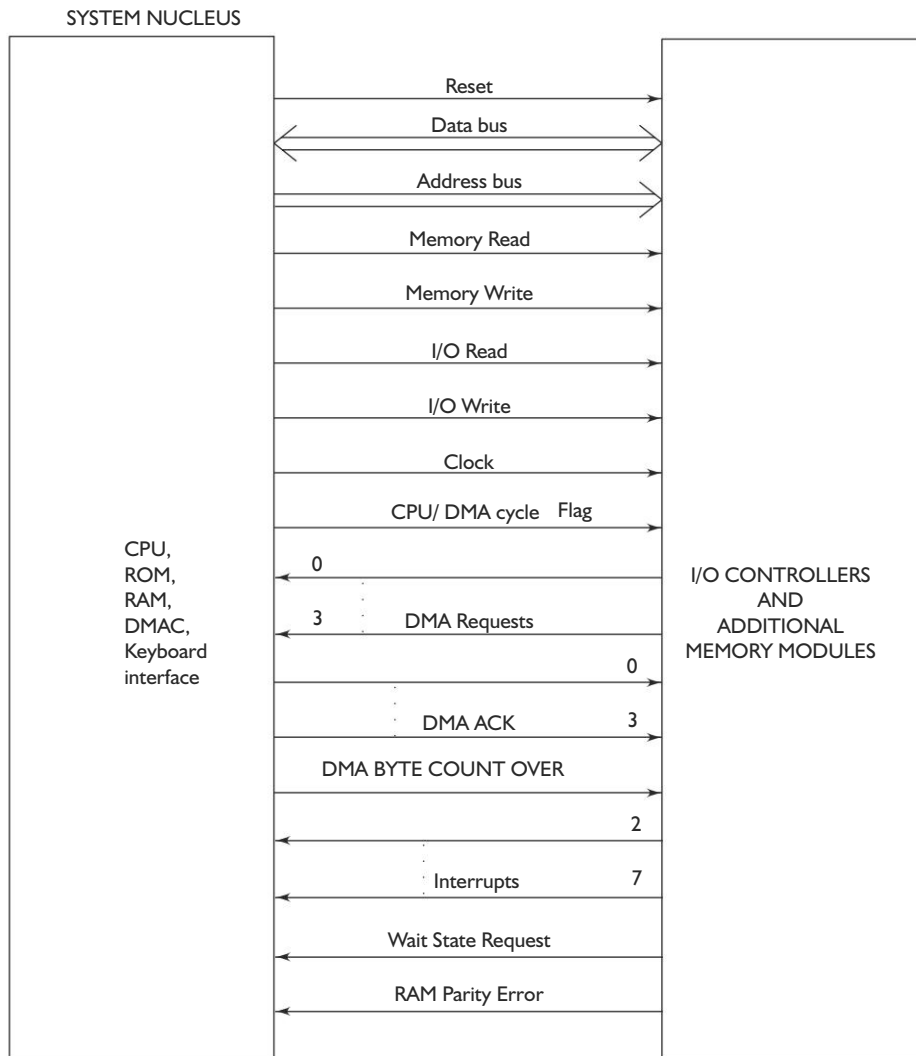
Fig. 10.73 8088-PC and system bus

internal systemboard subsystems (RAM, ROM and I/O chips) are connected to X bus (XA, XD, XC). The external daughterboards are connected via I/O slots which carry A, D and C busses. Thus X bus and D buses serve both as system bus and I/O bus without any differentiation. In this approach, CPU communicates with both memory and I/O subsystems with the same speed. Conceptually, there is only one bus as presented in Fig. 10.76. The separation into X bus and D bus are for electrical load distribution and physical modular interconnections between motherboard (system board) and expansion boards.

10.6.6.2 Isolated Memory and I/O Buses

The single bus concept is simpler to design and implement. But it has a drawback: limited bandwidth due to slow peripherals. This affects CPU – memory communication. This is never felt seriously, if the processor speed is less. For high speed processors, it is better to have separate memory bus and I/O bus as displayed in Fig. 10.77. The bandwidth of I/O

bus is less than that of memory bus. Hence, processor–memory communication takes place at a faster speed than both processor–I/O communication and memory–I/O communication. In addition, if external cache memory is used, it helps improving CPU performance.



Note: To simplify understanding, simple terms have been shown instead of actual signal names. Also the polarity information (Active low or high) is omitted.

Fig. 10.74 PC bus signals

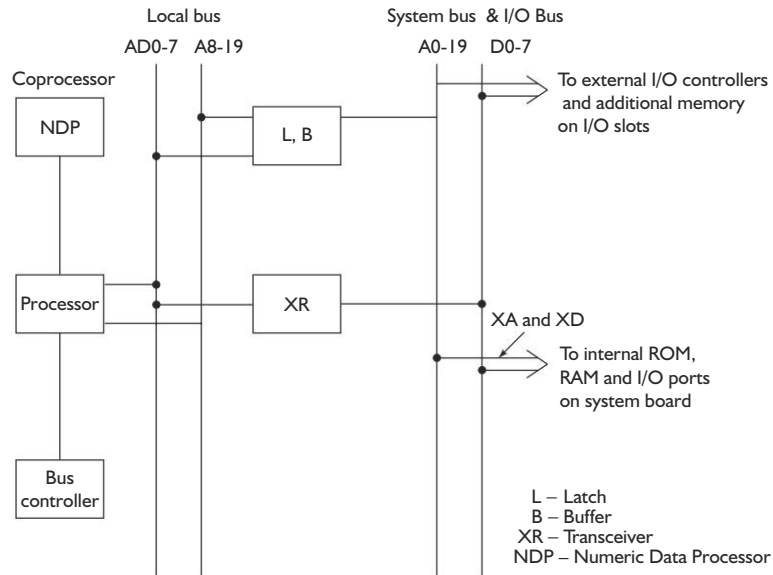


Fig. 10.75 Single bus structure in 8088-PC

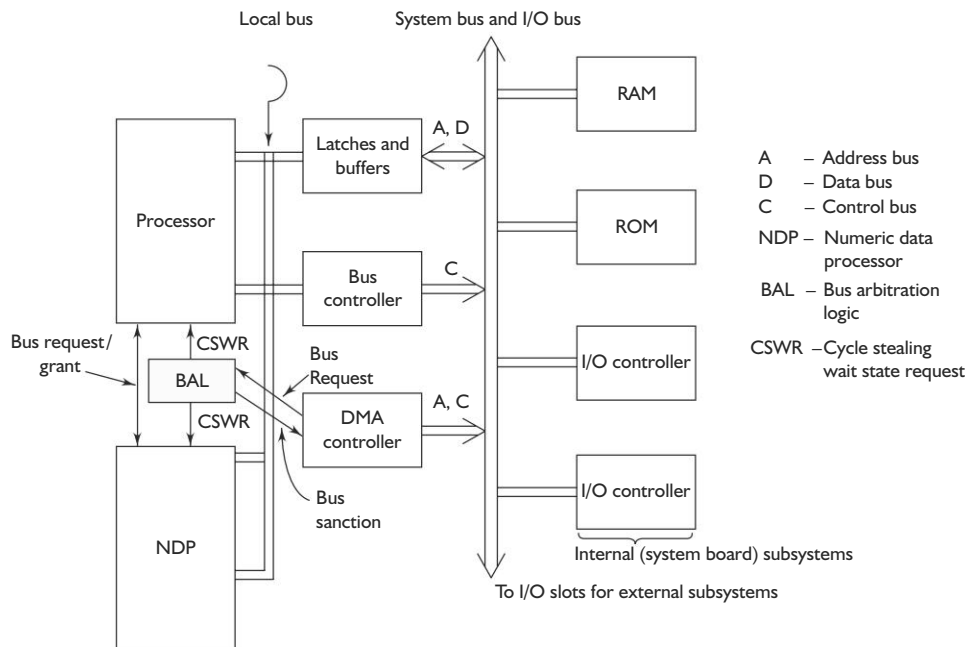


Fig. 10.76 Single bus concept in 8088-PC

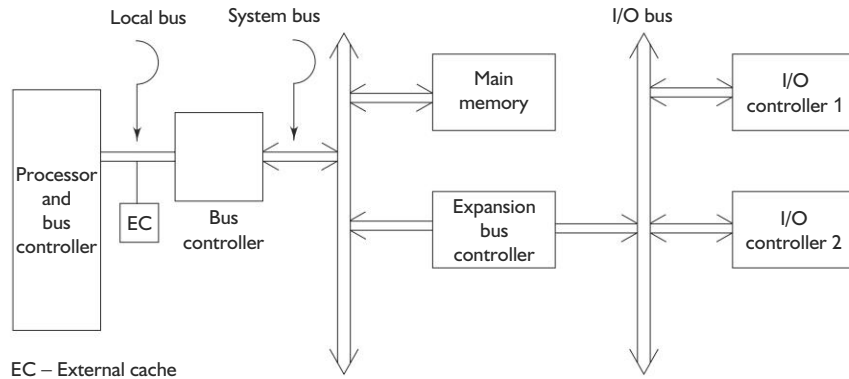


Fig. 10.77 Separate memory bus and I/O bus

10.6.6.3 Local Bus

There is need for supporting high speed peripherals such as fast hard disk drives, advanced display controllers, graphics processors etc. An old solution for this was connecting limited I/O controllers on local bus so that these controllers can communicate with memory at local bus speed. The local bus is a path allowing peripheral controllers to access main memory directly without going through the expansion I/O bus. Fig. 10.78(a) shows the concept of local bus. Usually, CRT controller is put on local bus slot. This 'short cut' enables higher data transfer for video controller, and hence better video performance. However, the local bus adapters may affect the CPU-to-memory traffic. Since the local bus adapters are electrically sitting directly at processor end, they communicate at a higher speed with memory. Each adapter should be designed carefully, so as not to load excessively the local bus

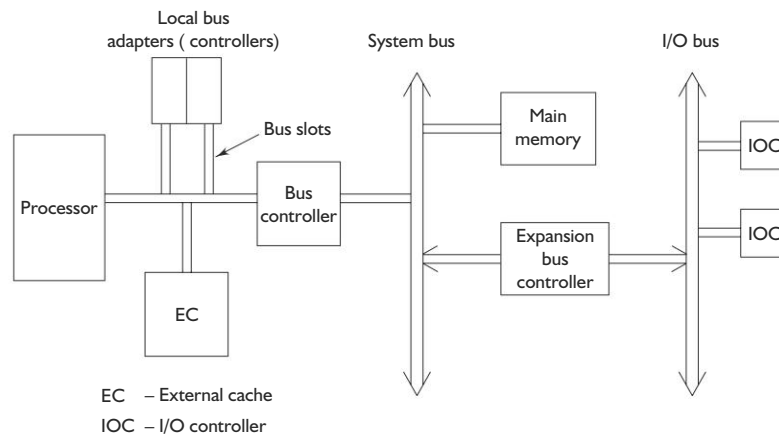


Fig. 10.78(a) Local Bus

signals. Any violation affects the functioning of the system. Too many adapters are not recommended. The VESA bus (VL bus) is a popular local bus standard widely used with 80486 processor based systems, to achieve faster data transfer with the CRT controller. It was announced by the Video Electronics Standard Association (VESA)

Bridging

A computer with multiple buses can support a wider variety of peripheral devices. One simple approach to implement multiple buses in a computer is using a bridge as illustrated in Fig.78(b). The bridge is a hardware circuit/device that interconnects two buses. The bridge acts as a translator between the two buses. It is allotted a range of addresses on each bus. When it is addressed on one bus, say bus A, it converts the address, and forwards the request to the other bus. The bridge *maps* the address space of bus A to the address space of bus B, and vice versa. The bridge should follow appropriate protocol on each bus.

Bridging permits a new computer design to provide access to one or more old type buses as auxiliary buses through the main bus of the computer. This allows us to interface all types of peripheral controllers/devices with different interfaces. Use of bridges in PC motherboard design is covered in next section.

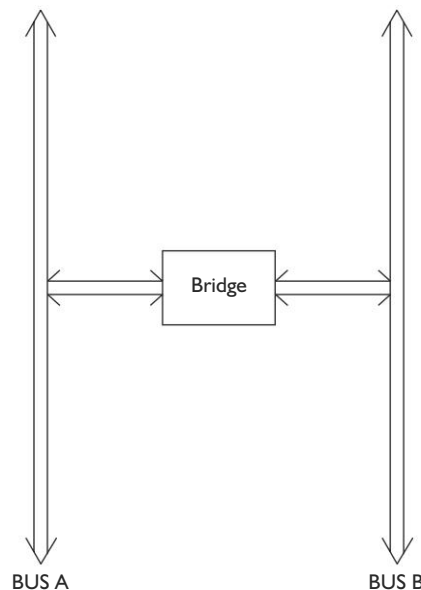
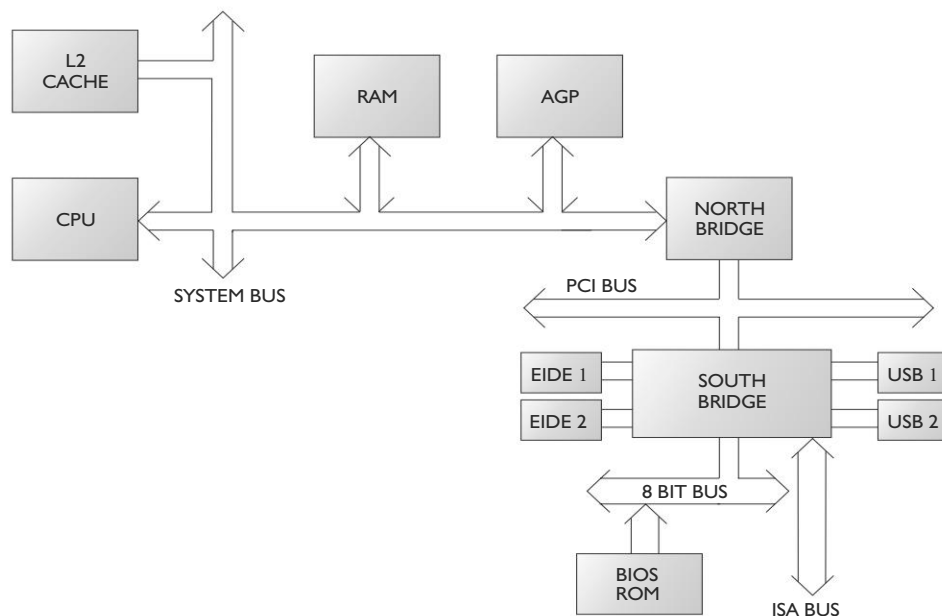


Fig. 10.78(b) A bridge linking two buses

10.6.6.4 PCI Bus and PC Motherboard

The local bus concept was soon replaced by Intel's PCI (Peripheral Component Interconnect) Bus. The PCI bus is not a real I/O bus. It is a mezzanine bus acting as an intermediate bus between the system bus and I/O bus. The PCI controller (bridge) links the system bus with PCI bus. The PCI bus isolates the slow speed I/O subsystems from the processor, but at the same time, the I/O controllers are brought closer to the memory. The PCI bus is a high speed bus. The I/O controllers on the PCI bus have another advantage: they can be used with any system, without bothering about the exact processor used, since they are not interfaced to the processor. Instead, they see only the PCI bus.

The structure of a modern PC motherboard design is based on a north and a south bridge. The bridges are *routers*; route data traffic from one bus to another. The north bridge handles the heavy traffic and the south bridge tackles a variety of narrow routes. Fig. 10.79 presents a block diagram of a modern PC motherboard organized, as a south bridge and a north bridge.



Note: The processor has on-chip L1 cache.

Fig. 10.79 PC chipset and bus on PC motherboard

The PCI bus is used for connecting I/O adapters, such as network-controllers, graphics cards, sound cards etc. The PCI functions like a 64 bit bus. The PCI bus provides buffering between CPU and I/O subsystems. Plug and Play (PnP) is an attractive PCI specification. All adapter cards for the PCI configure themselves. In a motherboard with PCI bus, usually there are two segments: the *Internal PCI bus* which connects to EIDE channels on the motherboard and the *PCI expansion bus* that extends on the I/O slots for PCI adapters. An advanced PC motherboard is architected similar to Fig. 10.80.

Figs. 10.81 and 10.82 illustrate multiple buses on a single computer. The local bus exists at the immediate input/output pins of the processor. The system bus is on the motherboard and it connects RAM to the processor. The I/O buses connect I/O subsystems to the processor. The I/O buses are generally derived from the system bus. The mezzanine bus is an intermediate bus between a local bus and I/O bus. It acts as a bridge between the I/O buses and the system bus. The PCI and multibus are examples of mezzanine bus. The architecture of PCI bus is discussed in section 10.8.

Table 10.10 lists certain high speed system buses.

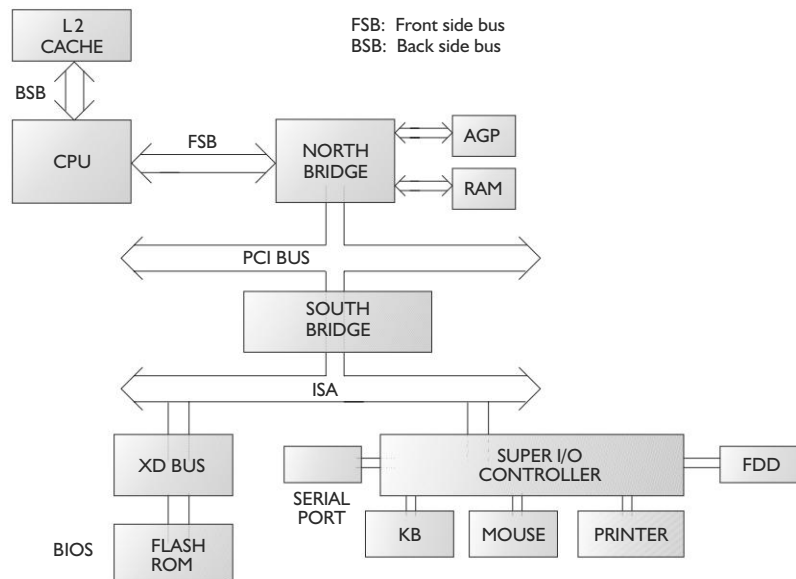
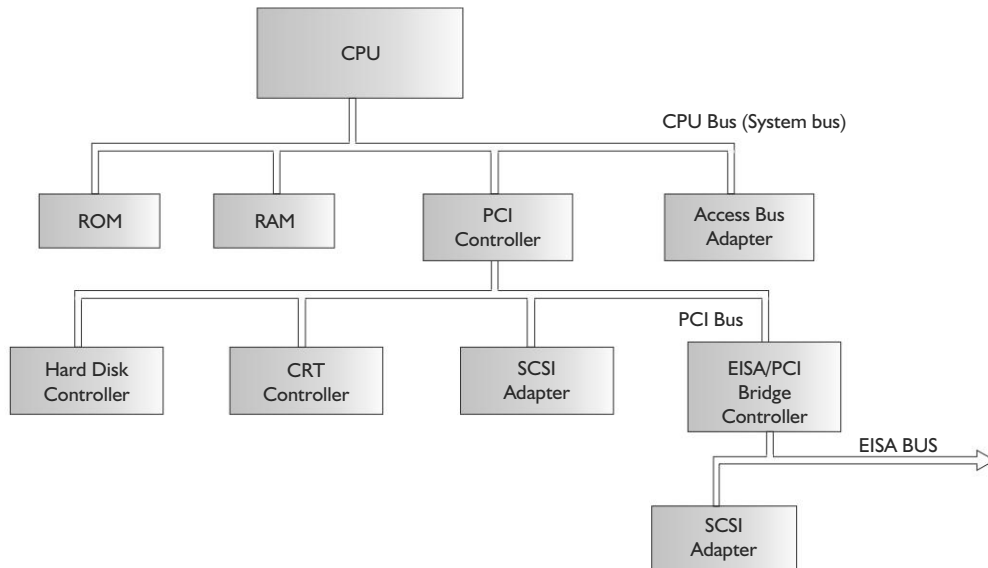


Fig. 10.80 Advanced PC motherboard and bus



Note: The processor and local bus are not shown explicitly

Fig. 10.81 System with multiple buses

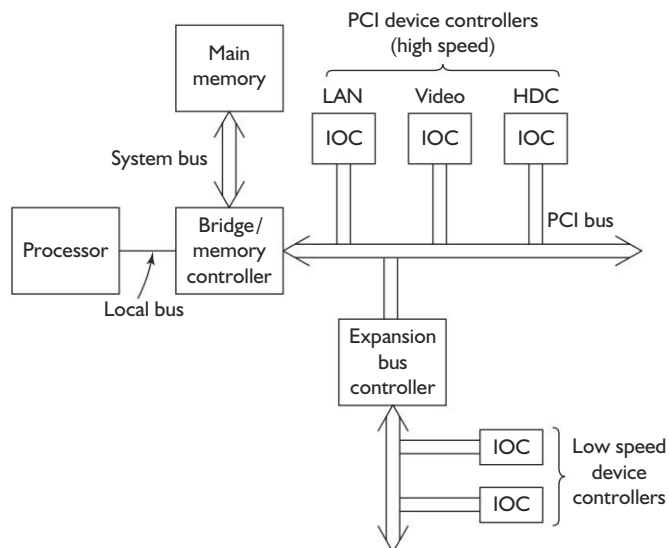


Fig. 10.82 PCI as a mezzanine bus

TABLE 10.10 High speed system buses

Processor	System bus speed (MHz)
Pentium II	100
AMD K6-2	100
Pentium II Xeon	100
Pentium III	133
AMD Athlon	200
Pentium 4	400/533

In advanced PCs based on 80386 and later processors, system bus and I/O bus are entirely separate and are operating at different speeds. In current PCs, we have multiple I/O buses. There are four common I/O buses present in current PCs:

1. ISA
2. PCI
3. USB
4. AGP

Table 10.11 compares the popular I/O bus types used in different models of PCs.

TABLE 10.11 I/O BUS standards in PCs

I/O bus	Bus width (bits)	Maximum transfer rate (MBps)
PC-8088 and XT	8	4-6
ISA (AT)	16	8
MCA (PS/2)	32	40
EISA	32	32
VL	32	100-160
PCI	32	132
USB	Serial	1.2
Firewire (IEEE1394)	Serial	80
USB 2.0	Serial	12 - 40

Example 10.2 A 32 bit PCI system operates at 33 MHz. Calculate the bandwidth.

Clock frequency = 33 MHz.

$$\text{Period} = 1/(33 \times 10^6)$$

One bus cycle transfers 4 bytes in one clock cycle.

$$\begin{aligned} \text{Hence, bandwidth} &= 4 \times 33 \times 10^6 \text{ bytes} \\ &= 133 \text{ MB/s.} \end{aligned}$$

10.7 Interface Circuits and I/O Controllers

The computer designers use a variety of techniques in the design of various I/O controllers. There are simple controllers which act more like a messenger between the software and the peripheral device. The right term for such a controller is the interface adapter. The printer controller is usually of this type. The other extreme is an intelligent controller, a highly sophisticated one with advanced features such as self-diagnostics, error retry, error correction, etc. Such a controller can execute complex commands. The hard disk controller is an example for intelligent controller. Fig. 10.83 shows the classification of device controllers.

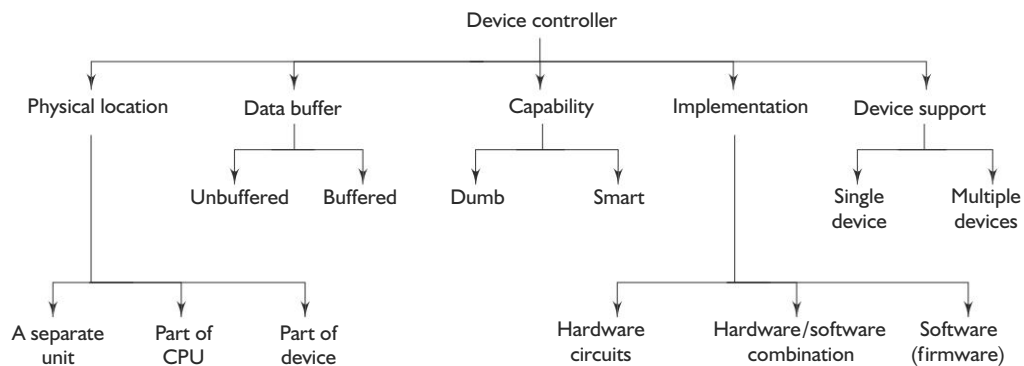


Fig. 10.83 Types of device controllers

10.7.1 Control Signals

Each control signal from the I/O controller, demands a particular control information/action at the device. There are basically three types of control signals:

1. Internal device action type
2. Data transfer type
3. Interface selection type

The controller generates appropriate control signals depending on the command received from the CPU/device driver. For example, to execute a seek command, the floppy disk controller issues a STEP signal to the floppy disk drive (Fig. 10.84). On receiving a STEP pulse, the disk drive moves the R/W head by one track. When the controller generates the MOTOR-ON signal, the disk drive rotates the



Fig. 10.84 Control signals

spindle (diskette) motor. To select a disk drive, the controller sends 'Drive select' signal, as a result of which the device is logically connected to the controller. The device recognizes any other signal only if it is logically connected to the controller. There may be several devices physically interfaced to a controller, but at a given instant only one of them is logically connected as shown in Fig. 10.85. The controller transmits the 'Drive select' signal to only that drive which is required in the current command. The data transfer control signal indicates to the device that now the controller is supplying data, in the case of output operation, or expecting data in the case of input operation. For example, the 'strobe' signal by the printer controller, reports to the printer that the controller has placed the data byte on data lines. The printer immediately receives the data. Table 10.12 defines the actions performed by the floppy disk drive for various control signals in the floppy interface.

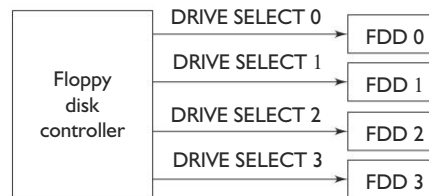


Fig. 10.85 Establishing logical connection

TABLE 10.12 Control signals in floppy interface

Sl. no.	Control signal	Action by the device
1	Head select	The FDD selects either top head (0) or bottom head (1) according to the state (0 or 1) of this signal
2	Step	The FDD moves the head to the adjacent track in the direction indicated by the direction control signal
3	Direction	This signal decides whether the FDD should move the head inward (toward the center) or outward (away from the center) for every step pulse
4	Write enable	The FDD supplies write current to the read/write head
5	Motor enable (motor on)	The FDD starts rotation of the spindle motor
6	Drive select	The FDD gets logically connected

10.7.2 Status Signals

Each status signal from the I/O device indicates a specific condition of the device. There are three types of status signals:

1. Interface selection type
2. Data transfer type
3. Internal status/condition type

**Fg. 10.86** Status signals

For example, the 'Track 0' signal is released by the disk drive (Fig. 10.86) when the Read/Write head is positioned above the track 0 (outermost track). The FAULT signal is generated by the floppy disk drive when there is an error condition in the drive. The 'SLCTD' from the printer indicates that it is logically connected to the controller. The ACK signal informs the controller that the printer has received the data byte transmitted by the controller. Table 10.13 defines the meaning of different status signals in the floppy interface.

The organisation of three different interface circuits of PC are discussed in the following sections, as case studies.

TABLE 10.13 Status signals in floppy interface

Sl. no.	Status signal	Meaning
1	Track 0	The read/write head is positioned over the track 0
2	Write protect	The diskette in the FDD is write protected
3	Ready	The FDD is ready for operation
4	Fault	There is an abnormal condition in the FDD; for example, both heads are selected simultaneously
5	Index	Index hole in the diskette is sensed; for each rotation, index hole is sensed once

Controller Hardware

There are different methods of designing an I/O controller hardware are as follows:

- (a) Hardwired design (random logic)
- (b) Special purpose LSI based design
- (c) Microprocessor based design
- (d) Custom built ICs or gate arrays

Hardwired Design

In this method, the controller hardware design consists of circuits using standard hardware components. Most of the old device controllers are of this type.

Special Purpose LSI Based Controller

In this method, a special programmable LSI IC designed exclusively for the specific I/O controller is used. The advantage of this method is the reduction in the controller hardware size, design time and the cost. In most microcomputers, the floppy disk controller is based on NEC 765 (or Intel 8272) which is a programmable floppy disk controller. The CRT controllers in early microcomputers make use of the 6845 which is a CRT controller IC.

The serial communications controller (asynchronous serial port) uses the 8250 which is an UART (Universal Asynchronous Receiver Transmitter) IC. The hard disk controller designs use the WD1010A which is a dedicated hard disk controller IC.

Microprocessor based Controller

In this method, the entire controller functions are achieved by programming a dedicated microprocessor. This leads to ease of design and reduction in hardware cost. The design involves developing certain routines to implement the controller functions. This method is usually followed where a large number of complex functions are executed by the I/O controller. However, the speed of operation of such a controller is less than the controllers of other methods. The keyboard interface in the IBM PC/AT is based on this principle. It uses the 8042 which is a microcontroller IC, a simple microprocessor with built-in memory and I/O ports.

Custom Built Controller ICs

In recent years, several controllers have been developed as special custom built ICs that incorporate the entire circuits required for the controller. These are known as ASIC (Application Specific Integrated Circuit) ICs. The advantage of this method is reduction in controller development time and controller size.

10.7.3 CASE STUDY 3: Keyboard Interface

The keyboard interface is one of the simplest I/O interfaces. The design strategy followed in personal computers uses a combination of hardware and software. The keyboard detects the key pressed and transmits a code (corresponding to the key) to the keyboard interface. The internal keyboard mechanism and code generation details are presented in Chapter 11. As of now, we assume the following behavior of keyboard, in personal computers:

1. *Serial Interface:* Fig. 10.87 shows the signals between the keyboard and CPU. The keyboard sends the serial data on the keyboard data (KB DATA) line. The keyboard also transmits the clock signal in order to provide timing information. Both the clock and data lines are bi-directional.

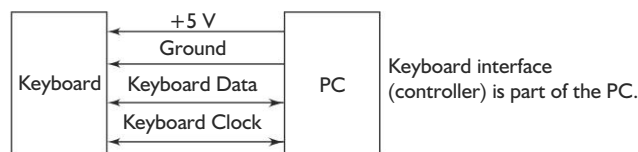


Fig. 10.87 Signals in the keyboard cable

2. *Scancode*: The keyboard uses 'scancode', an unique 8 bit pattern for each key. The keyboard sends two scancodes: when a key is pressed, the keyboard generates a make scancode; when the key is released, the keyboard generates a break scan code. The D7 bit is 0 in make scancode and 1 in break scancode. The remaining seven bits are same in make and break scancodes.
3. *Typematic action*: When a key remains pressed for more than 0.5 second, the keyboard repeats the make scancode, at a rate of 10 cps until the key is released.
4. *Internal buffer*: The keyboard has a 20 byte internal buffer to store the scancodes.
5. *Self-test*: The keyboard performs self-test when the CPU pulls the KBCLK line low. Then the keyboard transmits X 'AA' as scancode if no fault is detected. If any key is stuck, the keyboard transmits the make scancode of stuck key.

10.7.3.1 Keyboard Data Transfer

Figure 10.88 shows the block diagram of keyboard data transfer. The keyboard interface performs data transfer in interrupt mode. The SIPO converts the serial scancode into parallel, and then generates the interrupt signal IRQ 1. Immediately, the SIPO is frozen and the

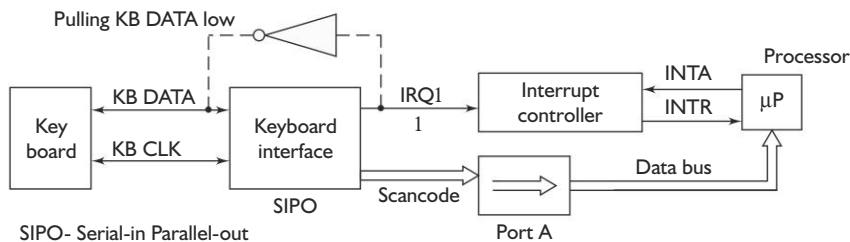
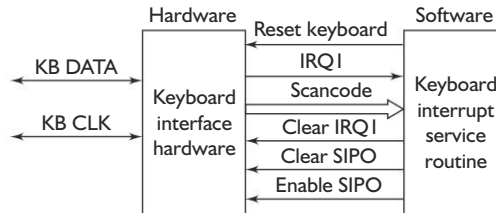


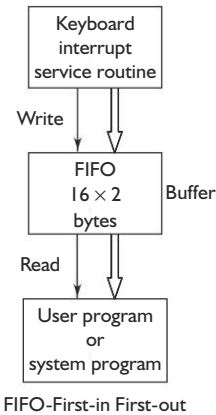
Fig. 10.88 Keyboard data transfer

KB DATA line is made low, indicating 'busy' (not yet over) condition to the keyboard. If required, the keyboard stores further scancodes temporarily in its internal buffer. The CPU branches to keyboard Interrupt Service Routine (ISR). The ISR reads the scancode (by an IN instruction) via port A. It also clears IRQ 1 and SIPO contents. The SIPO and keyboard data line are back to normal condition, to be ready for next scancode.

Figure 10.89 shows the communication between hardware and software. The ISR gives control to the keyboard driver that converts the scancode into ASCII format. The driver

**Fig. 10.89** Hardware-software communication

has a FIFO buffer in RAM as shown in Fig. 10.90. The driver stores both scancode and ASCII code in the FIFO buffer as shown in Fig. 10.91. In case, the ASCII pattern corresponds to a special (control) character, such as print screen pause, warm boot, etc., the driver takes appropriate action. Fig. 10.92 shows the actions executed by the I/O driver. The character code from the bottom of the FIFO buffer is read by the user program, through BIOS call, by means of software interrupt instruction (INT 16).

**Fig. 10.90** Use of a keyboard buffer

10.7.3.2 Keyboard Interface in Advanced PCs

Advanced PCs use single chip INTEL 8042/8742 as keyboard interface. This chip has following components:

- Built-in ROM (EPROM in 8742),
- RAM,
- I/O ports,
- Timer/counter,
- Clock generator,
- Status register,
- Two data registers.

With over 90 instructions and 18 programmable I/O pins, this chip is commonly used as an universal peripheral interface.

Scancode	ASCII pattern
----------	---------------

(a) Standard character key

Special key code	Zero
------------------	------

(b) Special key

Fig. 10.91 Format of keyboard buffer entry

The major functions performed by the keyboard interface chip are listed as follows:

1. Receiving serial data from keyboard.
2. Converting serial data into parallel scan code.
3. Checking for parity error.
4. Sending Resend command to keyboard on detecting an error in data.
5. Translating scan code into ASCII code.
6. Presenting the ASCII data to the processor through the output buffer.
7. Forming the status to indicate any error condition.
8. Raising interrupt to the processor.

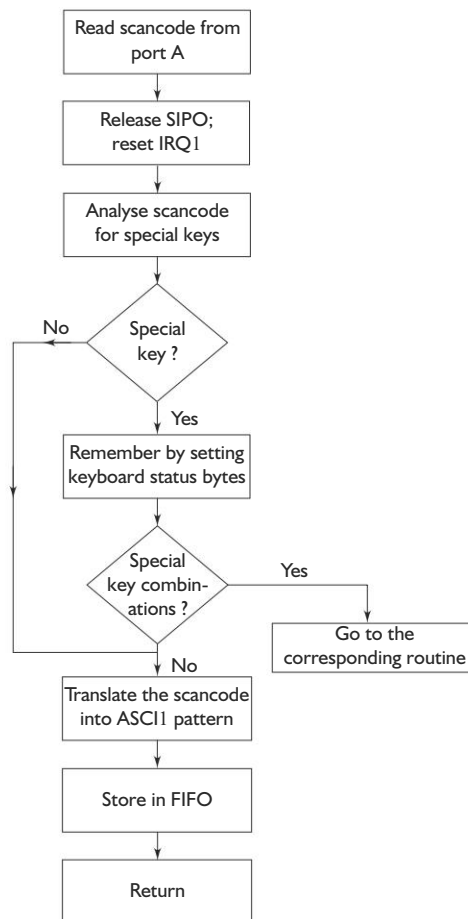


Fig. 10.92 Actions taken by the keyboard ISR

9. Receiving the data from the processor through the input buffer.
10. Generating parity bit.
11. Converting the data into serial.
12. Transmitting the serial data to the keyboard.
13. Receiving acknowledgement from the keyboard for each character released.

10.7.4 CASE STUDY 4: Centronics Interface Printer Controller

Traditionally, printer is interfaced to the CPU in two ways:

1. A serial interface printer via the serial interface of RS-232 C type.
2. A parallel interface printer via the Centronics interface.

Modern printers support interfacing to Universal Serial Bus (USB) which handles a variety of peripheral devices. The centronics interface is an uni-directional interface that is replaced by new bi-directional interfaces: Enhanced Parallel Port (EPP) and Extended Capability Port (ECP). These are discussed in section. 10.7.5.

Centronics Interface Signals: The centronics interface provides a handshake protocol between a computer and a printer. It supports a maximum data transfer speed of about 100 KB/s. Fig. 10.93 shows the signals in the centronics interface.

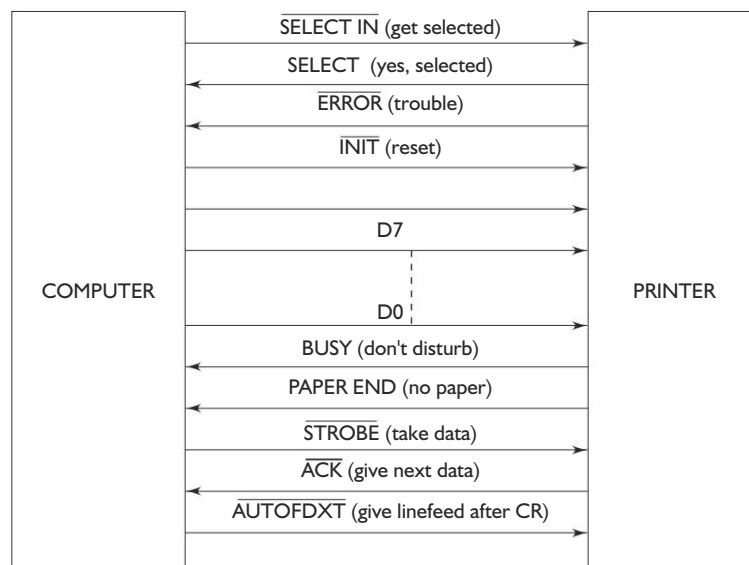


Fig. 10.93 Centronics interface signals

10.7.4.1 Signals from PC to Printer

There are 12 signals from the PC to the printer. Out of these, eight signals are data bits and four signals are control signals. The control signals are: $\overline{\text{STROBE}}$, $\overline{\text{INIT}}$, $\overline{\text{SLCTIN}}$, $\overline{\text{AUTO FEED XT}}$. All the control signals are low active. Their meanings are given below.

$\overline{\text{STROBE}}$: The printer should receive the data, when this signal is low.

$\overline{\text{INIT}}$: When $\overline{\text{INIT}}$ is low, the printer resets its electronics logic, and clears the printer buffer.

$\overline{\text{SLCTIN}}$: $\overline{\text{SLCTIN}}$ is an interface enable signal. When this signal is low, the printer responds to signals from the controller. Otherwise, the printer remains logically disconnected.

$\overline{\text{AUTO FEED XT}}$: After printing a line, the printer will provide one line feed automatically, if this signal is low. This type of line feed is known as hardware line feed.

10.7.4.2 Signals from Printer to PC

There are five status signals from the printer to the PC. These are $\overline{\text{ACK}}$, BUSY , PE , SLCT , and $\overline{\text{ERROR}}$.

$\overline{\text{ACK}}$: $\overline{\text{ACK}}$ signal is an acknowledgement for $\overline{\text{STROBE}}$ signal from the PC. When active, it indicates that printer has received data sent by the PC, and is ready to accept next byte.

BUSY : When BUSY signal is high, it indicates that the printer is busy, and it cannot receive data. This signal becomes high under any of the following four conditions:

1. On receiving $\overline{\text{STROBE}}$ active
2. During printing operation
3. When the printer is in offline state
4. When the printer senses some error condition

PE : When PE signal is high, it indicates that there is no paper in the printer. Either the paper is torn or the paper is over.

SLCT : SLCT signal indicates that the printer is selected and logically connected to the PC.

$\overline{\text{ERROR}}$: $\overline{\text{ERROR}}$ signal indicates that there is some error condition in the printer. The three reasons for generating this signal are:

1. Mechanical fault or electronic fault in the printer
2. The printer is in offline state
3. There is no paper in the printer, i.e., paper-end state

10.7.4.3 Timing Diagram

The timing diagram of Centronics interface protocol is illustrated in Fig. 10.94. The printer controller sends data to the printer. After a minimum gap of $0.5\ \mu\text{s}$, it makes $\overline{\text{STROBE}}$ low, and keeps it low for a minimum duration of $0.5\ \mu\text{s}$. As soon as $\overline{\text{STROBE}}$ becomes low, the printer makes the BUSY line high. The controller should retain data on the data

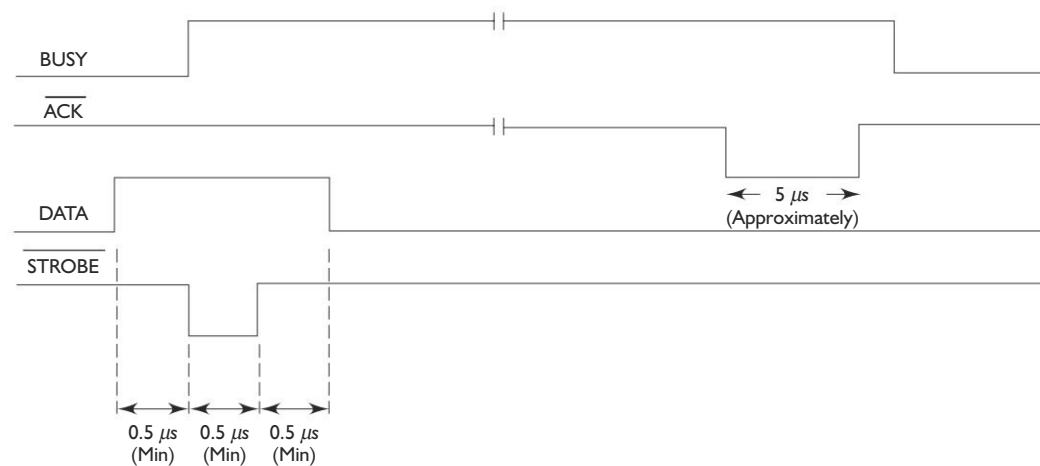


Fig. 10.94 Centronics interface timing diagram

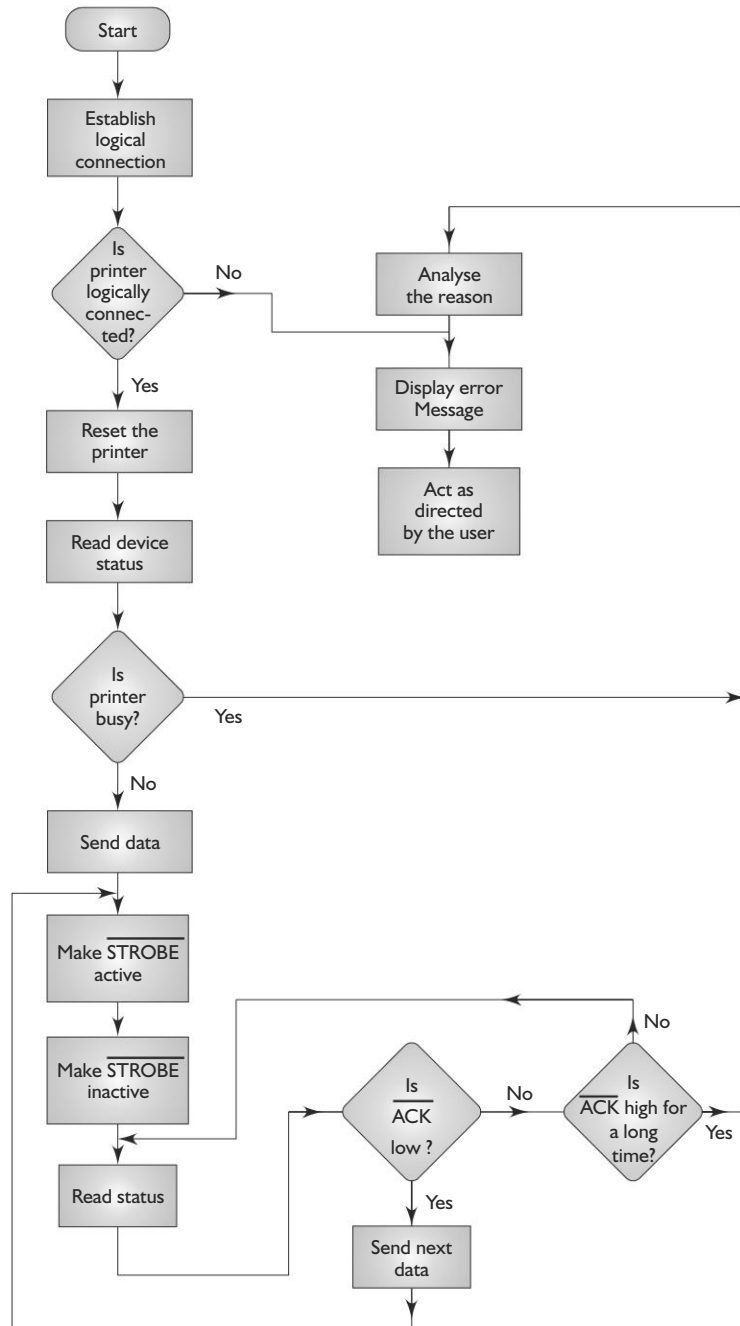
lines for a minimum interval of $0.5\ \mu\text{s}$ from the trailing edge of $\overline{\text{STROBE}}$. Thus the data should be kept on the data lines for a minimum duration of $1.5\ \mu\text{s}$. When the printer is ready to receive the next character of data, it makes $\overline{\text{ACK}}$ line low. When $\overline{\text{ACK}}$ is made inactive, the printer also removes BUSY.

10.7.4.4 Programming Sequence

Figure 10.95 presents a flow chart for the programming sequence.

1. As a first step, the software should verify the logical connection of the printer. For this, the PC should send $\overline{\text{SLCT IN}}$ signal to the printer. If the printer is powered-on, and in online mode, it will respond with SLCT signal. If the printer does not send a SLCT signal for a long time, the software should draw the attention of the operator by a message.

2. Before sending data, the printer should be in online mode, error-free and not busy. This can be verified by sensing at the BUSY signal.

**Fig. 10.95** Flowchart for printer routine

3. If the printer is free, the software can output the data character. After a minimum of 0.5 μ s delay after sending data, the $\overline{\text{STROBE}}$ signal should be issued to the printer. As soon as $\overline{\text{STROBE}}$ signal is received, the printer makes BUSY signal high, and it receives the data. The printer will analyse the data and determine whether it is a message character to be printed, or a control character communicating some control action to be done by the printer.

Some programs may cause hardware line feed by activating $\overline{\text{AUTO FEED XT}}$, and some programs may cause software line feed by sending the line feed control character. If both features are simultaneously used, it will result in double line spacing.

4. The printer controller generates an interrupt (IRQ7) to the CPU, on sensing a negative edge on $\overline{\text{ACK}}$ signal. Thus the printer routine can be interrupt driven.

The printer controller provides options for both program mode and interrupt mode. The software chooses the mode of data transfer, by sending IRQ Enable command to the printer controller.

In short, the software directly generates the control signals by setting the appropriate bits in the command port. Similarly, the status signals are directly sensed by software by reading the status port.

10.7.4.5 Loop Back

The printer controller provides loop back facility for both command and data ports. This is done by making these two ports, bidirectional. Hence the software, after sending one byte of data to the printer controller, can read the data back by IN instruction. The printer controller loops back the data. By comparing the data sent and the data returned, the software can determine whether any bit is lost or picked up. Similarly the command bits can also be read by the software by an IN instruction on the command port.

10.7.4.6 Data Buffer

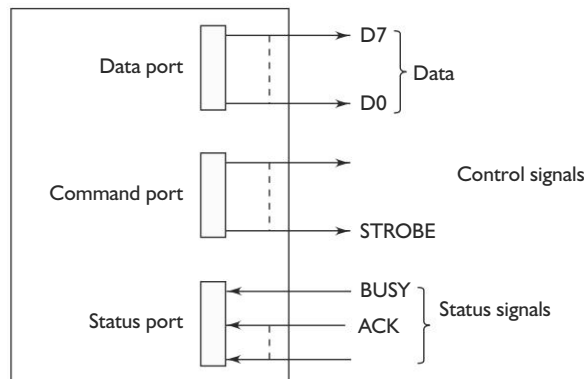
Usually, the printer has a buffer memory to store the characters received from the PC. Hence, the printer need not take action for the ASCII character immediately after receiving it. It simply stores the ASCII character in the buffer memory and immediately issues acknowledgement to the PC by sending $\overline{\text{ACK}}$. Hence, the data transfer through the centronics interface is done at a faster speed.

Some printers are designed with large buffer memory so that their overall through-put is quite outstanding. The printer simultaneously does both printing and receiving data. When it is printing one line, it can receive data for some subsequent lines.

10.7.4.7 Design Strategy

The printer controller merely acts as a messenger between the program and printer (Fig. 10.96). Observing the protocol sequence, and interpreting the status signals, are all done by the I/O driver. The input and output ports in the printer controller are the means through which the driver communicates with the printer. As shown in Fig. 10.97, there are three ports:

- Data port,
- Command port,
- Status port.

**Fig. 10.96** Interface adapter**Fig. 10.97** Ports in a printer controller

The *data port* and *command port* are output ports whereas the *status port* is an input port. Each port is assigned a unique address that is decoded by a common port address decoder as shown in Fig. 10.98. The data bus transceiver acts as a bi-directional communication path between the system data bus and the three ports. The bits in the command port are issued as the control signals. To issue a control signal to the printer, the driver uses an OUT instruction for control port address. The data pattern in the processor register is entered into the

command port. To read the status signals, the driver uses an IN instruction for the status port. The status signals are read through the status port, and stored inside processor register. To issue data to the printer, the driver uses an OUT instruction for the data port. The contents of processor register are transmitted on the data lines. The printer controller is capable of performing data transfer either in program mode or in interrupt mode as per the choice made by the I/O driver. The EI bit in the command port is used for this purpose as shown in Fig. 10.99. When $EI = 1$, the printer controller generates IRQ 7, as soon as the printer sends ACK signal. Thus after every byte is issued to printer, IRQ 7 is raised. When $EI = 0$, IRQ 7 is not generated. Hence, the status of ACK signal is sensed by the I/O driver, by looping on IN instruction for the status port, till ACK becomes active.

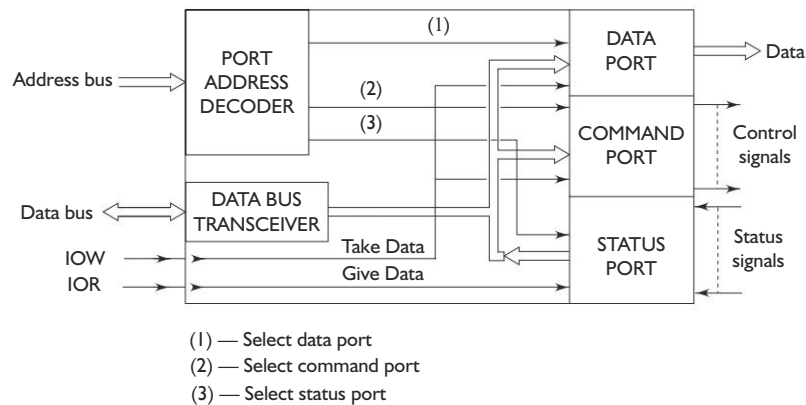


Fig. 10.98 Functional blocks in a printer controller

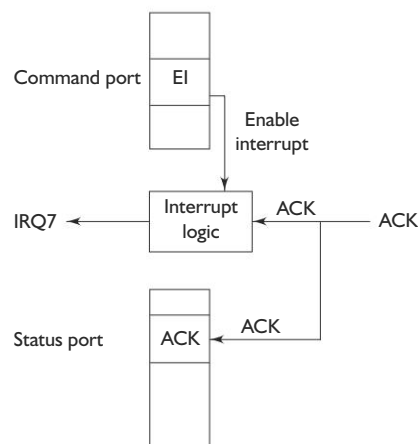


Fig. 10.99 Selecting mode of data transfer

10.7.5 New Generation Parallel Ports

A variety of new peripherals such as magnetic tape, portable disk drive, LAN adapter, CD ROM drive, printer sharer etc. can be easily interfaced to PC, via the parallel port. Since the new peripherals demand higher transfer speed and bi-directional transfer, the IEEE 1284 standard was introduced.

The IEEE Std.1284 defines a Bi-directional Parallel Peripheral Interface for PC. This standard enables high speed bi-directional communication between the PC and an external peripheral. It defines five modes out of which two are bi-directional, and others are uni-directional. In one of the modes, it supports the original standard parallel port (SPP) without any change. The different modes are listed in Table 10.14.

TABLE 10.14 IEEE 1284 parallel port modes

Sl. no.	Mode	Data-path (bits)	Direction	Protocol responsibility	Remarks
1	Compatibility mode (centronics/standard mode)	8	Forward (PC to peripheral)	CPU/BIOS (same as SPP)	This mode is mainly for backward compatibility to older peripherals; together with a reverse channel mode (nibble or byte), bi-directional transfer is possible
2	Nibble mode	4	Reverse (peripheral to PC)	CPU/BIOS	Four status lines from the peripheral are used to transfer the data from the peripheral; two cycles are needed for reading one byte of data
3	Byte mode	8	Reverse (peripheral to PC)	CPU/BIOS	Converts the data path in the SPP into bi-directional path by adding a provision to disable the interface drivers for the data lines
4	Enhanced parallel port (EPP)	8	Bi-directional	Controller hardware	Mainly for non-printer peripherals such as CD ROM, tape drive, hard disk drive, network adapter, etc.
5	Extended capability port (ECP)	8	Bi-directional	Controller hardware	This mode is mainly for new generation of printers and scanners

The first three modes, (Compatible, Nibble, and Byte), are based on programmed mode/ interrupt mode of data transfer. The I/O driver handles a handshake protocol similar to SPP and possible data transfer rate is up to 50–100 k bytes per second. The EPP and ECP modes are based on hardware method of data transfer. The software performs minimum operations such as initiation, and the printer controller handles the handshaking and data transfer without the involvement of the software.

A peripheral need not support all the modes. The 1284 standard defines a protocol for the PC to find the supported modes, and select the requested mode. These two actions are executed during ‘negotiation’ sequence defined by the IEEE 1284. When a PC performs a negotiation sequence, the old parallel port will remain silent, without sensing it. Hence, the PC decides to operate in compatibility mode which is similar to original Standard Parallel Port. A peripheral designed using IEEE 1284 standard responds to the negotiation sequence. The PC can ask the peripheral to transmit its Device ID or ask it to enter a specific mode.

10.7.5.1 EPP Mode

Table 10.15 shows the signals applicable to SPP and EPP modes. The EPP mode supports four different types of transfer cycles:

TABLE 10.15 Parallel port signals in SPP and EPP modes

SPP signal/ compatibility mode name	EPP mode signal name/function
$\overline{\text{STROBE}}$	$\overline{\text{WRITE}}$: Indicates a write operation
$\overline{\text{AUTO FEED}}$	$\overline{\text{DATA STB}}$: Data read or Data write operation is in progress
$\overline{\text{SELECT IN}}$	$\overline{\text{ADDRSTB}}$: Address read or Address write operation is in progress
$\overline{\text{INIT}}$	$\overline{\text{RESET}}$: Peripheral reset
$\overline{\text{ACK}}$	$\overline{\text{INTR}}$: Interrupt to PC
BUSY	$\overline{\text{WAIT}}$: LOW indicates its readiness for starting a new cycle; HIGH indicates its clearance to end the current cycle.
PE	User defined and device dependent.
SELECT	Same as above
$\overline{\text{ERROR}}$	Same as above
DATA (8:1):	AD (8:1): Bi-directional address/ data

- Address Write,
- Address Read,
- Data Write,
- Data Read.

The data cycles are used for sending data between the PC and the peripheral. Figure 10.100 shows a Data Write cycle. The address cycles are used for conveying one of the following: Address, Channel, Command or Control information. Figure 10.101 shows an Address Read cycle. A parallel port using EPP mode has additional ports than in SPP.

Table 10.16 lists the ports in an EPP controller. In EPP mode, the supply of data between the parallel port and the PC is carried out, by one OUT or IN instruction for the data port. The control ports and status ports provide backward compatibility. But the protocol sequence with the device is performed by the hardware in the parallel port, without the knowledge/involvement of the PC software/BIOS. Transfer rates in the order of 500 K– 2 M bytes per second is obtained by using the EPP mode.

The Data Write Cycle Sequence in EPP mode is given below.

1. The I/O driver issues an OUT instruction to port 4 (EPP Data Port). The CPU conducts an IOW write bus cycle.
2. The EPP controller activates $\overline{\text{WRITE}}$ signal and places the data byte on DATA lines.

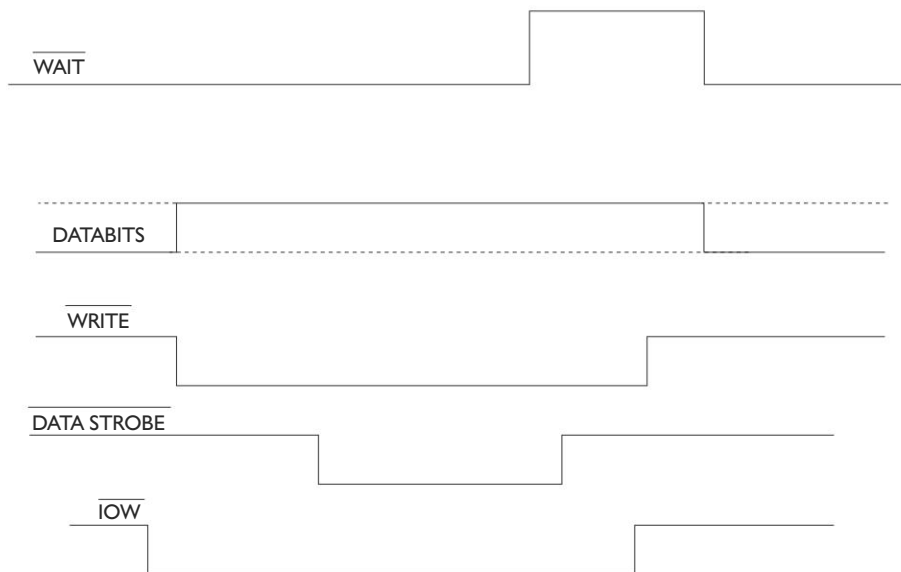
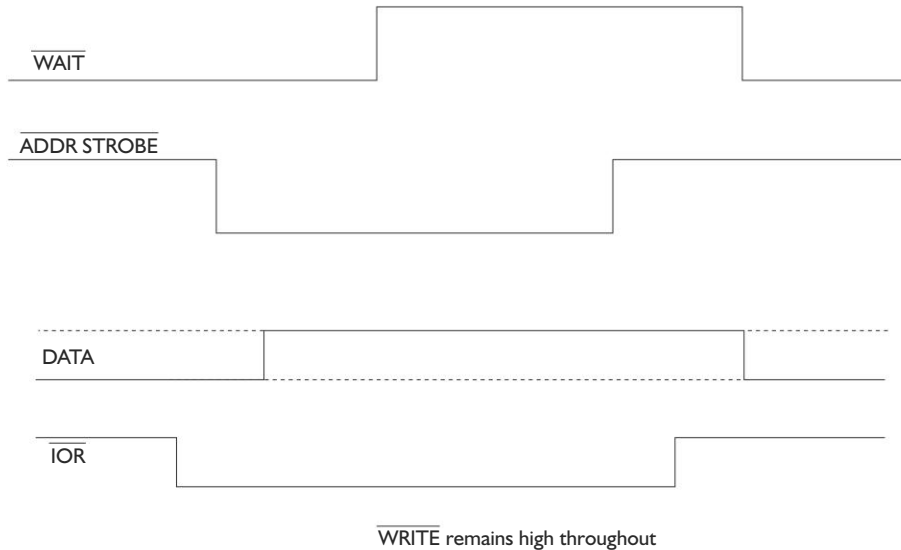


Fig. 10.100 EPP data write cycle

**Fig. 10.101** EPP address read cycle

3. The peripheral maintains the $\overline{\text{WAIT}}$ signal LOW.
4. The EPP controller activates the Data Strobe.
5. The EPP controller waits till the peripheral deactivates the WAIT.
6. The EPP controller deactivates the Data Strobe.
7. The peripheral makes $\overline{\text{WAIT}}$ LOW.

TABLE 10.16 EPP ports

Sl. no.	Names of port	Type	Remarks
1	Data port	O	Same as SPP data port.
2	Status port	I	Same as SPP status port. Normally not used by new systems.
3	Control port	O	Same as SPP control port. Not used by new systems.
4	EPP address port	I/O	Initiates an address read or write cycle.
5	EPP data port	I/O	Initiates an data read or write cycle.

Note: I—Input to CPU; O—output from CPU; I/O—input/output.

10.7.6 CASE STUDY 5: RS 232C Interface

The serial interface was used in early days more as a cost saving option for long distance communication. The RS232C standard is a popular serial interface.

10.7.6.1 Data Communication Fundamentals

The need to provide data transfer between a computer and a remote terminal has led to the development of serial communication. Serial communication implies transfer of data, bit by bit, on a single communication line, as shown in Fig. 10.102. The cost of the communication hardware is considerably reduced since only a single wire is required for the data bits.

When a digital signal is transmitted through a long wire or transmission line, the signal gets affected in two ways, as listed below:

1. The signal is attenuated, i.e., the voltage level drops due to the distributed resistance.
2. The transmission line has both distributed inductance and capacitance due to which the signal is distorted, i.e., the leading and trailing edges have slow rise and fall time.

The attenuation and distortion of the digital signal increases with length of the line.

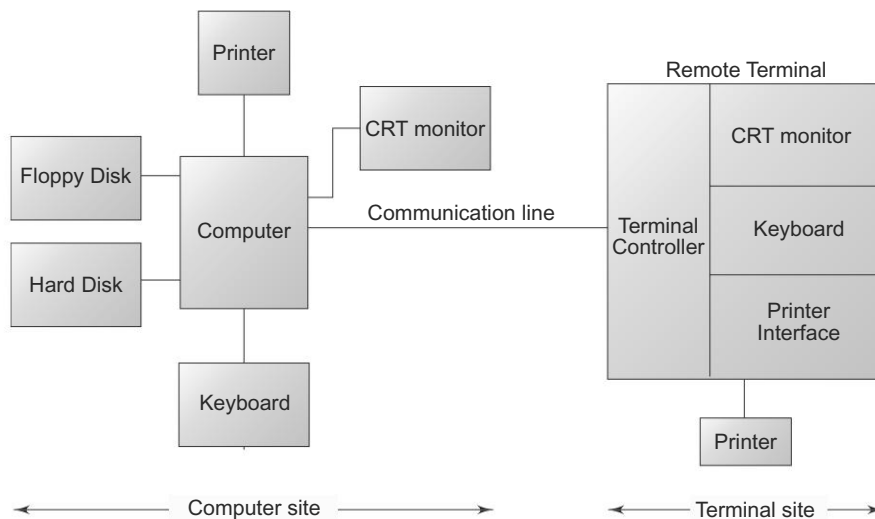


Fig. 10.102 Data communication to a remote terminal

Hence the signal cannot be transmitted effectively over long distances. To overcome this, a MODEM (MODulator-DEModulator), a communication equipment, is used for long distance data transfer through a telephone line. A pair of modems is necessary to link the two ends of communication.

Asynchronous and Synchronous Communication

Serial data communication generally uses either asynchronous or synchronous communication scheme. These two methods employ different techniques for synchronising the circuits in the sending end and the receiving end. In the asynchronous scheme, each character includes start and stop bits, as shown in Fig. 10.103. In the synchronous scheme, after a fixed number of data bytes, a special bit pattern called SYNC is sent as shown in Fig. 10.104. There can be gaps between adjacent characters in the asynchronous communication whereas there is no gap between adjacent characters in the synchronous communication. There is a continuous stream of data bits coming at a fixed speed in a synchronous communication scheme. The rate at which the data bits are sent is known as the Baud Rate, specified in BPS (Bits Per Second). In the asynchronous communication scheme, the bits within a character frame (including start, parity and stop bits) are sent at the baud rate.

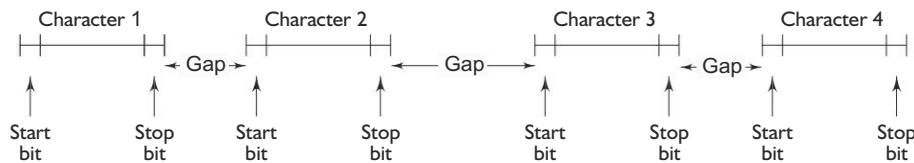


Fig. 10.103 Asynchronous communication

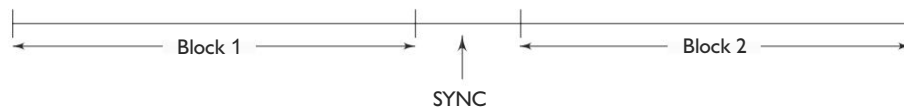


Fig. 10.104 Synchronous communication

Synchronous communication is used generally when two computers are communicating to each other or when a buffered terminal is communicating to the computer. Asynchronous communication is used when slow speed peripherals communicate with the computer.

The serial format of a character in asynchronous communication is shown in Fig. 10.105 in negative logic. The figure also shows the serial data format for the data A6. When there is no transmission, the line is maintained at a high voltage level called MARK. The start bit takes the line to a low level called SPACE and it indicates that one character transmission is about to start. The duration of the start bit is equal to one baud period.

The LSB follows the start bit and then the other bits follow in the order. The duration of each data bit is equal to one baud period. The line becomes high or low depending on the data bits. The number of data bits in a character can be from five to eight as desired by the transmitting and receiving ends.

The parity bit is sent after the most significant data bit. The parity bit may be odd or even. It is also possible to drop the parity bit. The duration of the parity bit is equal to one baud period.

The stop bit brings the line to a high (Mark) level and the stop bit duration could be 1, 1.5 or 2 baud periods. The number of stop bits is programmable. After the stop bits, the line remains in a high level (mark).

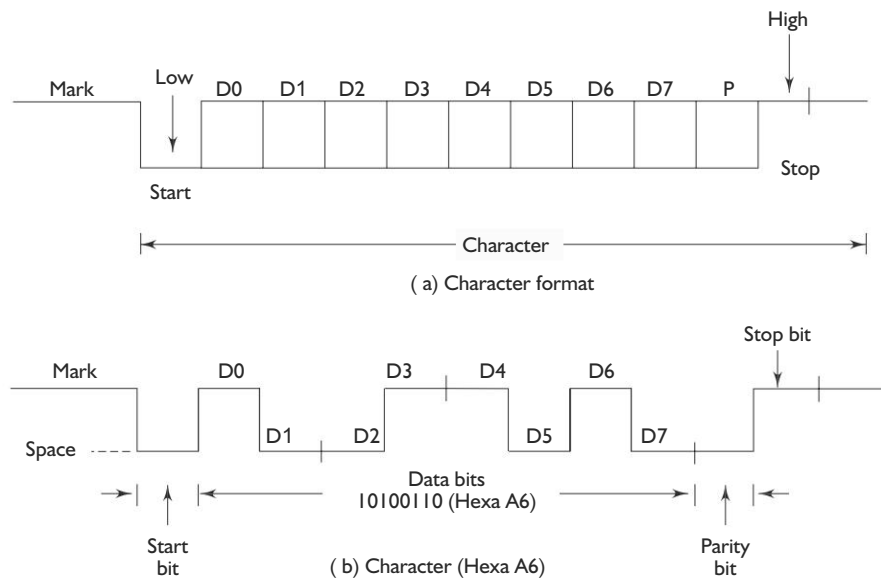


Fig. 10.105 Character format in asynchronous communication

The standard baud rates are: 50, 110, 134.5, 150, 300, 1200, 2400, 4800, 9600, 19200 and 38400. The baud rate chosen in a communication system depends on the quality of the transmission line and the capability of the transmitting and receiving end equipments. In a simple transmission system, the baud rate is same as the Bits Per Second (BPS), i.e. speed of data transfer and number of bits transmitted per second on the line are same. In communication systems where multiple modulation levels are used, the baud rate and the bits per second are not equal, i.e., the speed of data transfer is not equal to the number of bits transmitted per second on the line. Because of simultaneous transmission through multiple logical channels over a single medium, the bit rate is higher than the baud rate.

10.7.6.2 Serial Interface

The basic components of a serial communication system involving a computer and a terminal are shown in Fig. 10.106. The terminal can be replaced by another computer or a printer. By using a modem, the digital signal is converted into an analog signal. The telephone line is used for transmitting the analog signal. At the receiving end, the analog signal is reconverted into digital data. The modem may employ different modulation techniques: Amplitude Modulation (AM), Frequency Modulation (FM), Frequency Shift Key Modulation (FSK), etc. The FSK method is widely used in modems.

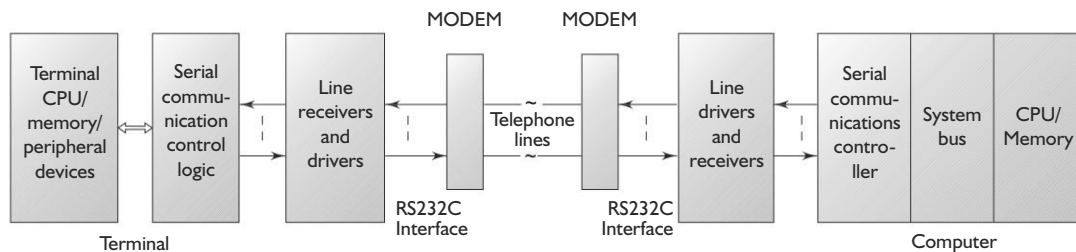


Fig. 10.106 Components of serial communication system

10.7.6.3 RS-232 Interface

The RS-232 interface is a standard interface specified by the Electronics Industries Association (EIA) (RS stands for Recommended Standard). The specifications of the RS-232C standard are similar to the CCITT V-5 standard recommendations.

The RS-232 interface expects a modem to be connected at the receiving and the transmitting end. The modem is DCE (Data Communication Equipment) and the computer, terminal, or printer with which the modem is interfaced is DTE (Data Terminal Equipment). The DCE and the DTE are linked via a cable whose length should not exceed 50 ft. Though not reliable, it may not affect the communication if the speed of data transfer is reduced when the distance is increased. The DTE has a 25 pin D type male connector and the DCE has a 25 pin D type female connector. However some systems use 9 pin connectors.

RS-232 Signal Levels

The RS-232 standard follows negative logic. A logical 1 is represented by a negative voltage and logical 0 is represented by a positive voltage. The level 1 (High) varies from -3 to -15 V and the level 0 (Low) varies from $+3$ to $+15$ V. In practice, the hardware

circuits used for the RS-232 interface maintain the signal level at + 12 V (logical 0) and at – 12 V (logical 1).

RS-232 Signals

Table 10.17 lists the RS-232 interface signals. The TXD carries the data bits sent by the DTE. The modem receives the TXD signal and uses it for modulating the carrier signal. The RXD is the data from the DCE to the DTE. The RXD is generated by the modem by demodulating the signal received from the other end modem.

Before sending data, the DTE requests for permission from the modem by sending the RTS signal. When the modem finds that the communication path (consisting of telephone line, the other end modem and DTE) is ready for communication, it issues the CTS signal to the DTE, as an acknowledgement for the RTS.

The DTE issues the DTR signal when it is powered-on, error-free and ready. The modem issues a DSR signal to indicate that it is powered-on and is error-free.

The RI and RLSD signals are used with the dialled modem. When the telephone line is a shared (switched) line, a dialled modem is used and a telephone set is attached to the modem. When a DTE at one end wants to communicate with a DTE at the other end, it initiates a dial sequence. The modem at the sending end sends a dial tone. In response, the called modem issues the RI Signal to its DTE, and sends an answer tone for 2s to the calling modem. Then, the calling modem sends an 8 ms duration tone on the telephone line. Now, the called modem issues CD to its DTE. The CD is an indication to the DTE that it will soon be receiving the data sent by the other end DTE.

TABLE 10.17 RS-232 Interface

Sl. no.	Signal	Signal name	Source	Destination
1.	—	Frame Ground	—	—
2.	TXD	Transmit Data	DTE	DCE
3.	RXD	Receive Data	DCE	DTE
4.	RTS	Request to Send	DTE	DCE
5.	CTS	Clear to Send	DCE	DTE
6.	DSR	Data Set Ready	DCE	DTE
7.	SG	Signal Ground	—	—
8.	RLSD	Received Line	DCE	DTE
	or	Signal Detect		
	CD	(Carrier Detect)		
9.	DTR	Data Terminal Ready	DTE	DCE
10.	RI	Ring Indicator	DCE	DTE

10.7.6.4 Serial Port in Original PC

The RS-232 standard interface is implemented in a wide variety of computer related equipment, such as terminal, printer, mouse, optical scanner, bar code reader, voice synthesiser, OMR (Optical Mark Reader), OCR (Optical Character Reader), process control systems, etc. These equipment are linked to the computer through modems when they are in a remote area. If the distance is considerably small, the terminal equipment can be directly

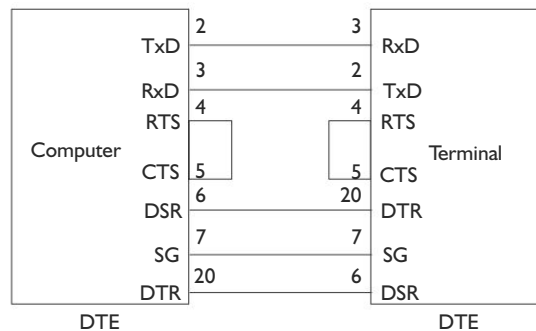
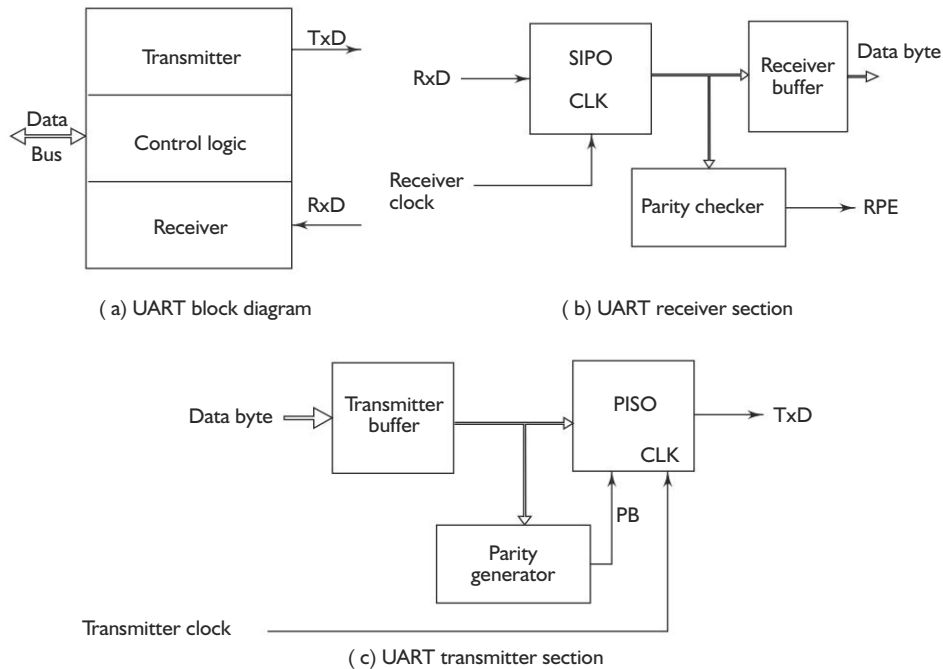


Fig. 10.107 A null modem linking two DTEs

connected to the computer provided appropriate modifications are done in the cable. Since both the computer and the terminal are DTEs, they have identical modem interface. As each end expects a modem (DCE) to be connected to it, the interconnections should cheat both the ends, as if each of them is linked to a modem. Such an interconnecting cable is known as a null modem and is shown in Fig. 10.107.

UART

The Universal Asynchronous Receiver Transmitter (UART) is a programmable LSI device having necessary hardware circuits for implementing asynchronous serial communication. Figure 10.108 shows the essential components of the receiver section and transmitter section in an UART. The receiver converts serial data bits received on the line into parallel bytes. The transmitter converts the parallel bytes into serial bits to be sent on the line. The frequency of the receiver clock has to match the baud rate of the receive data (RXD). The SIPO (Serial In Parallel Out) logic deserialises the serial data bits into a parallel byte. The RPE (Receiver Parity Error) signal is generated by the parity checker if there is a wrong parity in the received character. In the transmitter section, the PISO (Parallel In Serial Out) logic converts the parallel byte into a serial bit stream. It also adds start bit, parity bit and stop bit to the data.

**Fig. 10.108** UART block diagram

The frequencies of the transmit clock and the receive clock need not be equal. But the baud rate of the sending end transmitter should be equal to the baud rate of the receiving end receiver. If the baud rates are not equal, the receiver section will generate an RFE (Receiver Framing Error) signal. When the receiver section finds an invalid data format, it issues the RFE signal. The received data is invalid in one of the following cases:

- (a) The start bit is sensed but no stop bit is found after the data bits and the parity bit.
- (b) The start bit is sensed but its duration is less than a baud period.

If the receiver clock frequency and the data format are proper, and still the RFE is generated, there is either noise on the line, or a fault in the line or receiver circuits.

The UART samples the line condition at a fixed frequency which is 16 times the baud rate. To satisfy this, the clock inputs to the UART should be 16 times the desired baud rate.

Figure 10.109 gives the block diagram of a serial interface controller using the National Semiconductor 8250 UART.

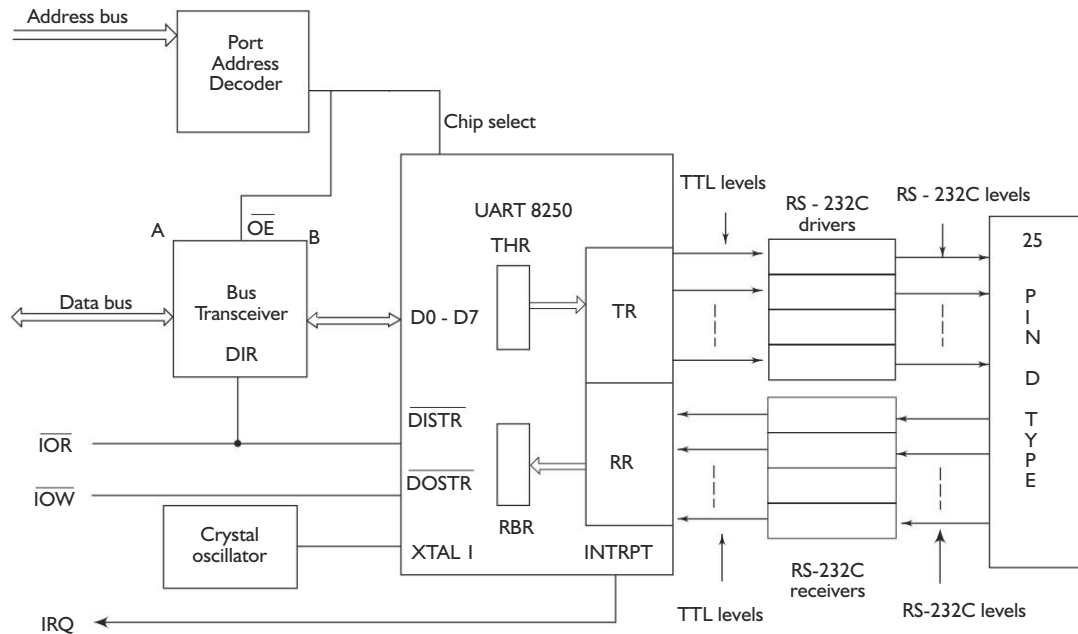


Fig. 10.109 Asynchronous communication port in PC

Internal Registers

The 8250 has 10 registers which are accessed by the CPU as I/O ports. Table 10.18 list the registers. The serial port driver accesses these registers by IN or OUT instructions.

TABLE 10.18 Internal registers

Sl.no.	Register	Remark
1.	Receiver Buffer (RB) or Transmitter Holding Register (THR)	During the input operation by CPU, the RB is accessed.
	Register (THR)	During the output operation, the THR is accessed.
2.	Interrupt Enable Register (IER)	Output register; four bits mask different interrupts.
3.	Interrupt Identification	Input register; indicates two types of information: (a) Interrupt Pending Status (b) Interrupt level which has been given priority.

(Contd.)

S.no.	Register	Remark
4.	Line Control	Input/Output register; stores format of character
5.	Modem Control	Controls the output signals to the modem; also provides loop back testing
6.	Line Status	Indicates different status conditions in the receiver and transmitter sections of the 8250
7.	Modem Status	Indicates the status of signals from the modem
8.	—	—
9.	Baud rate divisor (L Byte)	The program issues 16 bits.
10.	Baud rate divisor (H Byte)	The 8250 divides the clock input by this number to obtain $16 \times$ baud rate clock

10.7.7 Modern Serial I/O Interfaces

High performance and flexibility are the attractive features of two modern serial interfaces: Universal Serial Bus (USB) and Firewire. They also support a wide range of peripheral devices—from keyboard to digital camera.

10.7.7.1 Universal Serial Bus

The USB excels the RS232C serial port in several aspects: easy installation, faster transfer rate, simple cabling and multiple device connections. The features of USB are as follows:

1. *Multiple devices*: Upto 127 different devices can be connected on a single USB bus.
2. *Transfer rate*: The initial USB standard supported 12 Mbps transfer rate. The USB 2.0 supports higher rates.
3. *Support for wide range of peripherals*: Low bandwidth devices such as keyboard, mouse, joystick, game pad, floppy disk drive, zip drive, printer, scanner etc.
4. *Hub architecture*: Each device is connected to an USB hub. The USB hub is an intelligent unit interacting to the PC on one side and the peripheral devices on other sides. It is more like a multi “tiered star topology”. Hence, a single USB hub establishes presence of multiple USB devices (upto 127)
5. *Hot pluggability*: A USB device can be connected without powering-off a PC. The ‘Plug and Play’ feature in the BIOS and the USB devices take care of detection, device recognition and handling. The user is totally relieved of configuration procedures.

6. *Power allocation:* USB controller in PC detects the presence (attachment) or absence (detachment) of the USB devices, and allocates of appropriate levels of electrical power.
7. *Ease of installation:* A 4-pin cable carries signals as shown in Table 10.19

TABLE 10.19 USB cable

Sl. no.	Pin no.	Signal
1	1	Power
2	2	Signal -
3	3	Signal +
4	4	Ground

8. *Host centric:* The CPU/software initiates every transaction on the USB bus. Hence the overhead on the software increases when large number of peripherals, involving large number of transactions are connected.

A detailed coverage of USB is presented in section 10.8

10.7.7.2 Firewire (IEEE 1394)

The *Firewire* was initially introduced by Apple PC and subsequently standardized by IEEE 1394. It has become an attractive option for a variety of high speed peripherals including multimedia devices such as digital video camera. The firewire features are as follows:

1. Hot pluggability. This is similar to USB.
2. Multiple devices upto 63.
3. Snap connection: no need for device ID, Jumper, DIP switch, terminator etc., for device configuration.
4. Power sourcing.
5. Dynamic reconfiguration.
6. Higher speed: 400 Mbps; 30 times higher bandwidth than USB.
7. *Peer-to-Peer interface:* each device on the firewire bus forms a node unlike USB.
8. *Isochronous data transfers:* firewire supports isochronous data transfers which are suitable for digital video and other time-critical media. The isochronous data transfer guarantees timely delivery of data between the nodes. The device, once connected to the firewire bus, grabs an allocated portion of the bandwidth so as to guarantee the timely delivery of its data. The bus automatically allocates 10 MB/sec for serial command overhead, and the rest for the devices. Once the firewire bus bandwidth is exhausted (fully allocated), it does not recognize the remaining devices.

9. *DMA support*: the firewire, unlike USB and IDE, supports DMA transfers. Firewire is well suited for the following devices:
- Digital camera
 - Scanner
 - Hard disk drive
 - Removable disk drives
 - Printer
 - Tape back up
 - Video tapes
 - Video disks
 - Set-up boxes
 - Music systems



10.8 Modern Standard I/O Interfaces

A large number of I/O interfaces have been developed by several computer / peripheral manufacturers for interfacing various peripheral devices. SA-450 interface for floppy disk drive, ST-506 interface for hard disk, IDE interface for hard disk, Centronics parallel interface for printer, Dataproducts parallel interface for printer, RS-232 interface for serial communication, and SCSI for hard disk, printer and magnetic tape are some popular I/O interfaces used by several computer/peripheral manufacturers in the past. In order to support a variety of new peripherals, and also offer higher performance, new standards have been introduced regularly. PCI, SCSI and USB are three different I/O interfaces used in modern computers. The concepts and performance details of these three I/O interfaces are discussed in the following sections.

10.8.1 PCI Bus

Two main differences among the several types of bus are the number of bits that can be transmitted at a time, and the operating frequency used. Currently, the two fastest types of PC expansion bus are the PCI and the AGP. The PCI bus is a high performance bus widely used, from embedded systems to enterprise servers. The PCI bus supports higher speed devices/applications such as audio, streaming video, interactive gaming, modems, etc.

10.8.1.1 PCI Bus History

PCI bus was developed by Intel to replace the Industry Standard Architecture (ISA) and Enhanced Industry Standard Architecture (EISA) buses. The ISA bus, a 16-bit bus, was

slow and supported only few types of devices. EISA is a 32-bit version of ISA. IBM's Micro Channel Architecture (MCA) bus, Apple's NuBus, Sun's Sbus, Amiga's Zorro II and Zorro III and VESA Local bus are some popular higher-performance buses. PCI was first used in personal computers in 1994 with Intel 80486 processor. With introduction of Pentium processor, PCI replaced earlier bus architectures such as EISA, VL, and Micro Channel. Subsequently, Apple, Motorola and others also used PCI in their designs. Later, Compaq, Hewlett-Packard, and IBM brought out "PCI-X" to increase speed. The PCI-X bus is an extension of the PCI bus designed to the market segment of network servers. PCI-Express, a new standard, offers increased performance by using a switch instead of the multi-drop bus.

The PCI bus was released by Intel in June, 1992. Since then, almost all PC expansion peripherals, such as hard disks, sound cards, LAN cards, and video cards have been using the PCI bus. But its maximum transfer rate of 133 MB/s was insufficient for modern 3D applications. As a solution to this problem, Intel created a new bus, called AGP, to increase the transfer rate of video via the AGP bus, which is faster. This, also reduced the traffic in the PCI bus.

10.8.1.2 PCI Bus Overview

PCI bus has become very popular because of several attractive features. These are discussed below.

1. *Speed:* PCI bus provides extremely high speed transfers. The initial version itself supported data transfers up to 132 MB/sec.
2. *Burst mode:* A burst of data means a series of words. PCI bus supports burst mode of data transfer: a series of words from consecutive locations in memory are transferred via the bus without disconnection. This is ideal for processors with on-chip cache memory, since multiple words from consecutive locations in memory are moved to cache memory during cache fill operations.
3. *PCI Bridge:* In view of the high speed data transfers, the PCI bus limits the number of expansion slots to 3 or 4. To add more expansion slots, PCI-to-PCI bridges are used as shown in Fig.10.110. PCI-to-PCI Bridges are ASICs that electrically isolate two PCI buses while allowing bus transfers from one bus to another. Each bridge has a primary PCI bus and a secondary PCI bus, each of which is electrically isolated from the other. Multiple bridges are cascaded to design a system with many PCI buses. The bridge enables bus transfers to be forwarded from one bus to another until the target/destination is reached.
4. *PCI advantages:* The PCI bus has following major advantages over other bus architectures:

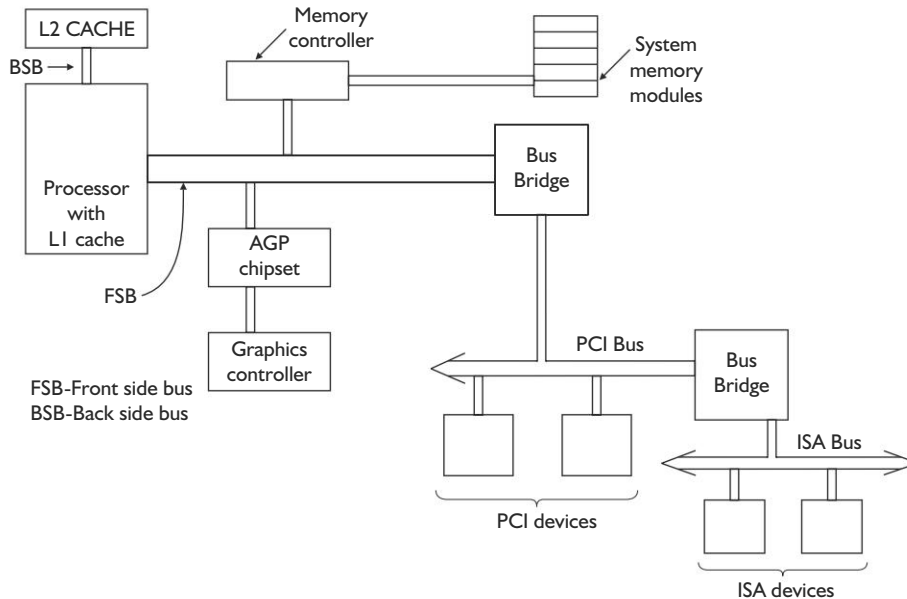


Fig. 10.110 A typical system with PCI bus

- a. PCI devices can have direct access to system memory without involving the CPU.
 - b. A single PCI bus can have up to five devices connected to it.
 - c. PCI supports bridges to handle large number of devices.
 - d. PCI supports auto configuration.
 - e. PCI bus is processor independent and hence can be used with any processor.
5. *Voltage Requirements:*
 The original PCI used 5V power. Subsequent PCI versions supported both 5 Volt and 3.3 Volt. The PCI 2.3 supports only the 3.3V power supply.
6. *PCI Variants and Clock speed:*

The initial PCI supported maximum clock rate of 33 MHz. At 33 MHz, a 32-bit slot gives a maximum transfer rate of 132 MBytes/sec, and a 64-bit slot gives 264 MBytes/sec. PCI revision 2.1 supported maximum clock at 66 MHz. The original PCI bus is 32 bits wide. This gives a transfer capacity of 132 megabytes/second. This is insufficient for gigabit Ethernet card, disk servers and database servers. To support these, one alternative is to use faster or wider variants: 66 MHz and 64-bit wide PCI bus. This increases the transfer to 533 megabytes/second. AGP (Accelerated Graphics Port) is based on PCI at 66 MHz and doubles the speed by performing data transfer on both edges of the clock. PCI-X (Peripheral Component Interconnect Extended) provides higher clock speeds and ECC for error checking and correction.

The PCI-SIG has introduced the PCI-X 2.0 standard to support faster transfers with new devices, and provides backwards compatibility with older devices. The newer standards such as Hyper Transport and PCI Express support better performance.

Table 10.20 compares the performance of different PCI bus versions.

TABLE 10.20 Performance comparison of PCI bus versions

Sl. no.	Bus	Clock speed	Bus width (in bits)	Data words per Clock Cycle	Maximum transfer rate
1	PCI	33 MHz	32	1	133 MB/s
2	PCI	66 MHz	32	1	266 MB/s
3	PCI	33 MHz	64	1	266 MB/s
4	PCI	66 MHz	64	1	533 MB/s
5	PCI-X 64	66 MHz	64	1	533 MB/s
6	PCI-X 133	133 MHz	64	1	1.066 GB/s
7	PCI-X 266	133 MHz	64	2	2.132 GB/s
8	PCI-X 533	133 MHz	64	4	4.266 GB/s
9	AGP x1	66 MHz	32	1	266 MB/s
10	AGP x2	66 MHz	32	2	533 MB/s
11	AGP x4	66 MHz	32	4	1.066 GB/s
12	AGP x8	66 MHz	32	8	2.133 GB/s

10.8.1.3 PCI Bus Features

PCI follows synchronous bus architecture; data transfers are done with reference to a system clock (CLK). Major features of PCI bus are briefly discussed below.

Bus Mastering and Bus Arbitration

The arbitration mechanism allows any device to request control of the processor bus. PCI has an arbitrator who can grant the use of the bus. Any device can get control of the bus. Devices use available bandwidth; a device can get even full bandwidth if no other demands are present. Initiators (bus masters) arbitrate for ownership of the bus by issuing a REQ# signal to a central arbiter. The arbiter grants ownership of the bus by activating the GNT# signal. REQ# and GNT# are unique for every slot. The current initiator's bus transfers are overlapped with the arbitration for the next owner of the bus.

Plug and Play and Auto Configuration

Plug and play feature allows addition of a device without need for any manual configuration. Setting jumpers and switches to configure address and interrupt level, IRQ, (like in ISA) is not required. PCI allots every device, a vendor and product ID, both 16-bit num-

bers. For instance, Intel's vendor ID is 0x8086. Some companies may have more than one vendor ID. Devices themselves identify broadly the type of function they perform such as storage, networking etc with a 16-bit class code; the first 8 bits indicate an overall class, such as mass storage or display; the second 8 bits can indicate subclasses, such as "SCSI" or "IDE" for mass storage, or "Ethernet" or "Token Ring" for network devices.

PCI supports an auto configuration mechanism. Each device has a set of configuration registers to identify the type of device (SCSI, video, Ethernet, etc.) and the manufacturer. Other registers enable configuration of the device's I/O addresses, memory addresses, interrupt levels, etc. The system configures itself; the PCI BIOS access the configuration registers at boot-up time. These registers indicate the resources required such as I/O space, memory space, interrupts, etc. The system allocates its resources accordingly, eliminating conflict among devices. A device's I/O address and interrupt level are not constant, and can change every time the system boots.

Multiplexed Bus

PCI uses multiplexed Address and Data bus. Both address and data are sent on the same set of pins at different times. Multiplexing reduces the pin count. This reduces the cost and size for PCI components.

Initiator and Target

In PCI terminology, data is transferred between an initiator (bus master), and a target (bus slave).

Address Phase and Data Phase

A bus transfer consists of one address phase and one or more data phases. A bus cycle is initiated with the address phase. The next clock edge starts one or more data phases. I/O operations have a single data phase. Memory transfers consist of multiple data phases.

Termination

Either the initiator or target may terminate a bus transfer sequence at any time.

10.8.1.4 PCI Bus Signals

Figure 10.111 shows the PCI bus signals. A brief description of major signals is given below.

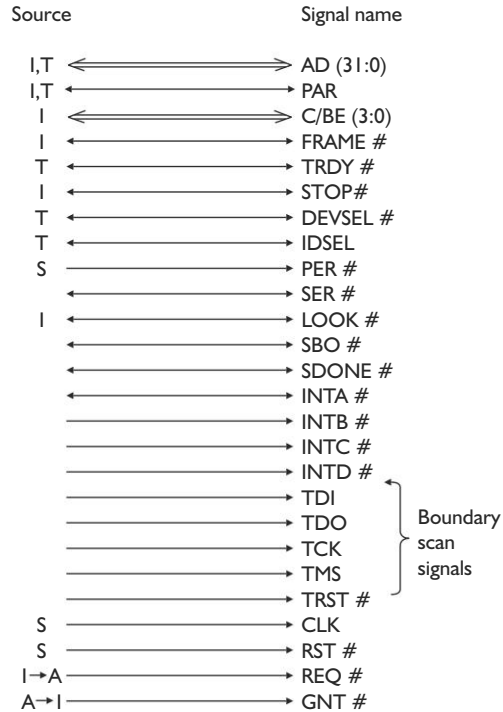


Fig. 10.111(a) PCI bus-basic signals

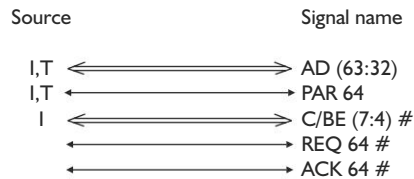


Fig. 10.111(b) PCI-64-bit optional signals

Note:

1. I - Initiator
2. T - Target
3. S - System
4. A - Arbiter

System Signals

CLK (Clock): Provides the timing reference on the bus for all transfers on the bus.

RST# (Reset): Resets devices; initializes configuration registers and output signals.

Address and Data Signals

AD [31:0]: Multiplexed Address and Data; carries a physical address during address phase, and contains data during data phase.

C/BE [3:0] #: Multiplexed Bus Command and Byte Enables; indicate the 'bus command' during the address phase, and the 'byte positions to be enabled', during the data phase.

PAR: Parity is even parity for the AD and C/BE# signals.

Interface Control Signals

FRAME#: Cycle Frame; Activated by the initiator in the start of a transaction, and indicate the duration of a bus transaction. In case of a single data phase, initiator will deactivate FRAME# after one cycle. In case of multiple data phases, the initiator will keep FRAME# active in all except the last data phase.

IRDY#: Initiator Ready; indicates that the initiator is ready to complete the current data phase. It means either the initiator has placed data, or the initiator is ready to accept data.

TRDY#: Target ready; indicates that the target is ready to complete the current data phase. It means either the target is ready to accept data, or the target has placed data.

STOP#: Stop; Target requests the initiator to terminate the current transaction. In the event of a fatal error such as a severe hardware problem, the target performs an abnormal termination of the bus transfer (target abort), by issuing STOP# and DEVSEL#.

LOCK#: Lock; Establishes exclusive access for the initiator while performing multiple transactions with a target. It prevents other initiators from modifying the address. LOCK# is mainly used by bridges to prevent deadlocks.

IDSEL: Initialization Device Select; Used as 'chip select' during configuration read and write transactions. IDSEL is unique for each slot and activated by the PCI system.

DEVSEL#: Device Select; Issued by a target when it detects its address on the bus.

Arbitration Signals (Initiator Only)

REQ#: Request; Issued by a device to the arbiter to request use of the bus.

GNT#: Grant; Issued by the arbiter to indicate that a device's request to use the bus has been granted.

Error Reporting Signals

PERR#: Parity Error; reports data parity errors during transactions.

SERR#: System Error; reports address parity errors, data parity errors during a special cycle, or any other fatal system error.

Interrupt Signals

INTA#, INTB#, INTC#, INTD#: Interrupts are issued by devices to request attention. Four level-sensitive interrupts are provided; each can be assigned to 16 separate devices. A PCI device that contains a single function uses only INTA#. Multi-function devices (such as a combination LAN/modem add-on board) use multiple INTx # lines. For example, a two function device uses INTA# and INTB#, etc.

Cache Support Signals (Optional)

These signals transfer status information between the bridge/cache and the target.

SBO#: Snoop Back off; indicates a *hit* to a modified line.

SDONE: Snoop done; indicates the status of the snoop for the current access.

Additional Signals

PRSENT [1:2] #: Present; Used for two purposes: 1) to indicate that an add-on board is physically present, and 2) to indicate the power requirements of an add-on board.

CLKRUN#: Clock Running; Optional signal to stop the CLK signal (when unnecessary) for power saving; relevant for the mobile environment where power consumption is critical.

M66EN: 66MHZ enable; Used in add-on boards that can support a 66 MHz CLK, and disabled in add-on boards that support only a 33 MHz CLK.

64-Bit Bus Extension signals (Optional)

AD [63:32]: 32 additional 'Address' and 'Data' in a 64-bit bus.

C/BE [7:4] #: 4 additional Bus Command and Byte Enables in a 64-bit bus environment.

REQ64#: Request 64-bit Transfer; Initiator indicates that it needs a 64-bit transfer.

ACK64#: Acknowledge 64-bit Transfer; Target indicates that it has decoded its address, and is capable of performing a 64-bit transfer.

PAR64: Parity Upper DWORD; Even parity bit for AD [63:32] and C/BE [7:4] #.

JTAG/Boundary Scan Pins (Optional)

PCI devices optionally support IEEE Standard JTAG/Boundary Scan. This provision helps easy automation of post production testing of microprocessors and other components. The components mounted on a PCI add-on board can be extensively tested by serially supplying test patterns through each component. There are five signals: TCK (Test Clock), TDI (Test Data Input), TDO (Test Output), TMS (Test Mode Select) and TRST# (Test Reset). TMS signal is used to set the test mode. TCK is a test clock which clocks the test data input, TDI, into the component. The TDI is a known data pattern. In test mode, a component such as the processor behaves in a special way, and not in its usual functional role. It propagates the TDI through various circuits inside the chip, and sends it out as TDO, test data output. Comparison of TDO pattern with expected result pattern, confirms whether the chip under test is good or bad.

10.8.1.5 PCI Bus Cycle Sequence

In a bus cycle, the bus master initially activates the FRAME # signal to indicate the commencement of the cycle; it then gives appropriate C/BE # pattern corresponding to the type of transfer and sends on the A/D lines, the address to be read from or written to. After the address and the command are transferred, the bus master activates the IRDY# signal to indicate its readiness to receive or send data. The target device responds with DEVSEL # to indicate that it has been addressed, and with TRDY# to indicate readiness to send or receive data. When both the initiator and the target are ready, one unit of data is transferred each clock cycle.

The initiator issues a command pattern on the C/BE [3:0] # during the address phase to indicate the type of transfer (memory read, memory write, I/O read, I/O write, etc). During data phase, the C/BE [3:0] # serve as byte enable to indicate which data bytes are valid. Data transfers occur on each clock edge in which both IRDY# and TRDY# are active. The initiator indicates completion of the bus transfer by deactivating the FRAME# signal. A target may terminate a bus transfer by activating the STOP# signal.

Dual Address Cycles

The PCI supports 64-bit addressing by performing two address phases. The low order 32-bits of the address are sent during the first address phase, and the high order 32-bits of the address during a second address phase.

Burst Access

In burst access, the PCI bus transfers a block of data from consecutive addresses. Instead of giving an address for each item of the block, the start address is given once at the beginning, and the block of data is transferred in a burst. The PCI bus transfers multiple units of data in frames. A frame begins with carrying an address and a command that indicates what type of transfer is done within the frame. Next, data is transferred in bursts of minimum duration.

Read Transaction

Figure 10.112 illustrates a read transaction on the PCI bus. A memory read operation by the processor, is considered here. It has been assumed that initially the bus is in idle condition.

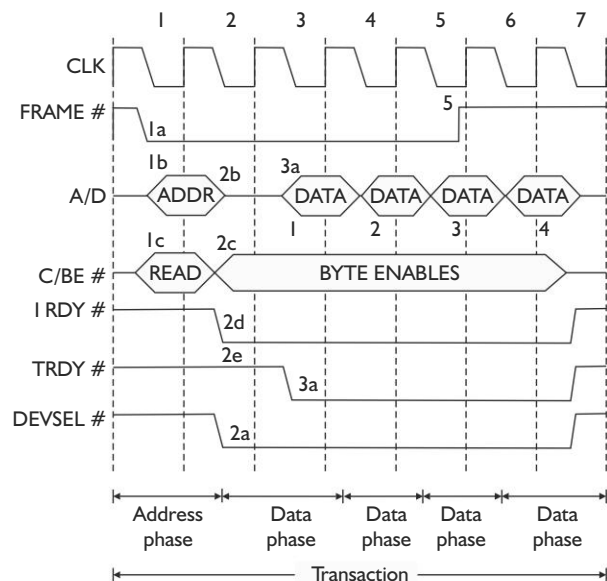


Fig. 10.112 PCI Bus read cycle

Cycle 1 – The initiator (in this case, the processor) does following actions: (a) activates the FRAME# signal to indicate the beginning of a bus transaction. (b) Places memory address (c) places the read command on the C/BE# signals. This is the address phase.

Cycle 2 – (a) The target activates DEVSEL# (in this cycle, or in the next) as an acknowledgment that it has decoded the address. (b) The initiator tri-states the address lines as preparation for the target to place data. (c) The initiator now issues byte enables information on the C/BE# signals. Each bit enables corresponding byte on the data lines. (d) The initiator activates IRDY# indicating it is ready to accept data. (e) The target deactivates TRDY# indicating it has not yet supplied valid data.

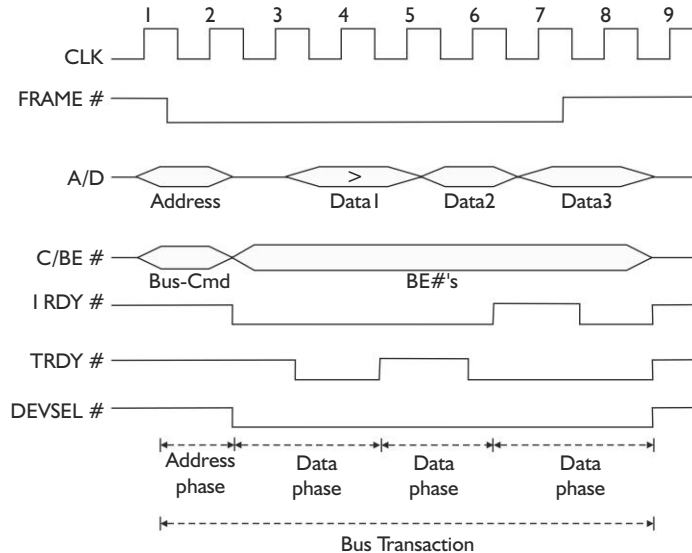
Cycle 3 – (a) The target places data and activates TRDY# indicating that data is valid. (b) IRDY# and TRDY# are both low during this cycle causing a data transfer to take place. (c) The initiator receives the data. This is the first data phase.

Cycles 4 – (a) the second data phase occurs since both IRDY# and TRDY# are active. (b) The initiator accepts the data provided by the target.

Cycle 5 and Cycle 6 – The target supplies two more words of data (similar to cycle 4) and these are captured by the initiator. The initiator deactivates FRAME# in cycle 5 (when it receives the third word) indicating that the next data phase is the final data phase.

Cycle 7 – FRAME#, AD, and C/BE# are tri-stated, and IRDY#, TRDY#, and DEVSEL# are deactivated for one cycle and then tri-stated.

The above sequence assumes that the target (in this case, memory) is fast enough to perform one word of data transfer in every clock cycle. In case, the target is not ready with data in any clock, it can deactivate TRDY#, indicating it needs more time to supply the next data word. In such a case, the initiator will not read the data lines until the target reactivates the TRDY#. Usually, this may result in one or more additional clock cycles for the data phase. Another special situation occurs due to delays caused by the initiator. Suppose in a data phase, the target has placed the data, but the initiator is not ready to receive it. In such a case, the initiator deactivates the IRDY#. In response, the target retains the data word on the AD lines, until the IRDY# is again activated by the initiator indicating its readiness. Figure 10.113 shows relevant timing diagram.



1. After supplying first word, the target is not ready with second word. Hence it deactivates TRDY# in clock cycle 4. In cycle 5, it sends second word and activates TRDY#
2. In cycle 6, the target sends third word. But the initiator is not ready to receive it. Hence the initiator deactivates IRDY#. As a result, the target retains the third word until the IRDY# is activated again.

Fig. 10.113 PCI read with pause/wait

Write Transaction

Figure 10.114 illustrates a write transaction on the PCI bus. The following is a typical sequence for a write transaction. A memory write operation by the processor is considered here. It has been assumed that initially the bus is in idle condition

Cycle 1 – (a) The initiator (in this case, the processor) activates the FRAME# signal to indicate the beginning of a bus transaction. (b) The initiator (processor) places the memory address and also places the write command on the C/BE# signals. This is the address phase.

Cycle 2 – (a) The target activates DEVSEL# (in this cycle or the next) as an acknowledgment that it has decoded the address. (b) The initiator places data on AD lines (c) The initiator now issues byte enable information on the C/BE# signals. Each bit enables corresponding byte on the data lines. (d) The initiator activates IRDY# indicating that valid write data is available. (e) The target activates TRDY# indicating it is ready to accept data. The

first data phase occurs as both IRDY# and TRDY# are active. The target captures the write data.

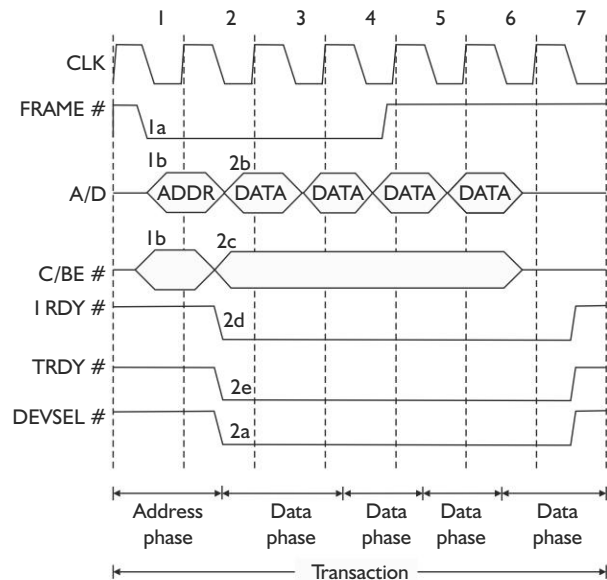


Fig. 10.114 PCI bus write transaction

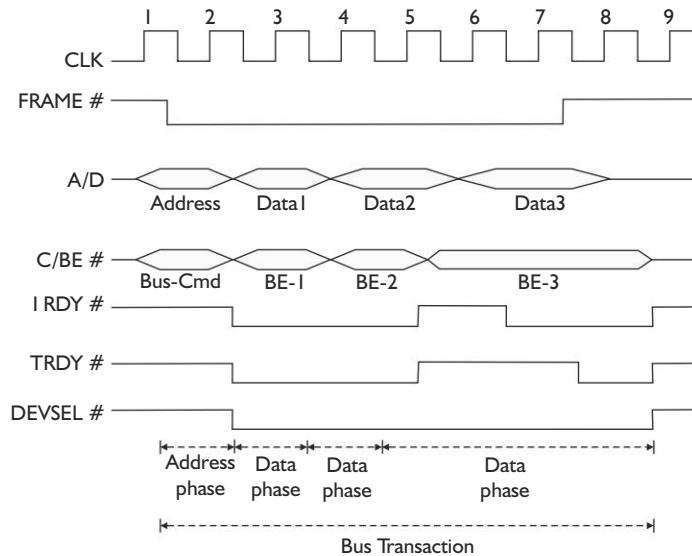
Cycle 3 – (a). The initiator supplies new data and byte enables corresponding to the second word. (b). The second data phase occurs since both IRDY# and TRDY# are low. (c) The target captures the data.

Cycles 4, and 5 – (a) The initiator supplies two more words of data (similar to clock cycle 3) and these are captured by the target. (b) The initiator deactivates FRAME# in clock cycle 4 (when it sends the third word) indicating that the next data phase is the final data phase (master termination).

Cycle 6 – FRAME#, AD, and C/BE# are tri-stated, and IRDY#, TRDY#, and DEVSEL# are deactivated for one cycle and then tri-stated.

The above sequence assumes that the target (in this case, memory) as well as the initiator are fast enough to perform one word of data transfer in every clock cycle. In case, the initiator is not ready to provide data, it can deactivate IRDY# indicating it is not ready to provide the next data. In such a case, the target will wait, until the initiator activates IRDY# again. Similarly if the target is not ready to accept data in any clock cycle, it can deactivate

TRDY# indicating it needs more time to accept the next data. In such a case, the initiator will not place the new data until the target reactivates the TRDY#. Usually this may result in one or more additional clock cycles for the data phase. Figure 10.115 illustrates this situation.



1. After supplying second word in third clock, the initiator is not ready for sending third word in next clock. Hence, the initiator deactivates the IRDY# in fourth clock cycle. It retains the second word. In fifth clock, it supplies third word and activates IRDY#.
2. Since the target is not ready to accept data in fifth and sixth cycles, the transfer takes place in seventh cycle.

Fig. 10.115 PCI write with pause/wait

Configuration Cycle

The PCI configuration mechanism can address each PCI device in the system. A PCI device is selected by a configuration cycle only if its IDSEL is active and following signal status is met:

- (a) AD [1:0] is “00” (indicating a type 0 configuration cycle),
- (b) Command on the C/BE [3:0] # signals during the address phase correspond to either “configuration read” or “configuration write”.

During the configuration cycle, the AD [10:8] may be used to select the desired function from the PCI device. AD [7:2] select individual configuration registers within a device and function.

10.8.1.6 PCI-X

The PCI-X is a high-performance enhancement to the PCI bus specification. The initial version of PCI-X doubled the maximum clock frequency from 66 MHz to 133 MHz, thus enabling transfer speeds over 1 Giga Byte/sec. It also improved the efficiency of the PCI bus and the devices attached to it, with new features such as split transactions and transaction byte counts. PCI-X was developed for applications such as Gigabit Ethernet, Fiber Channel and other Enterprise server applications.

PCI-X 2.0 increased speed and performance even further. Maximum clock frequencies of 266 MHz and 533 MHz are now supported to permit fast transfers of up to 4.3 Gigabytes per second of bandwidth. PCI-X 2.0 also adds additional features for high reliable subsystems such as RAID, Fiber Channel, InfiniBand{tm}, and iSCSI.

PCI for Data Communication

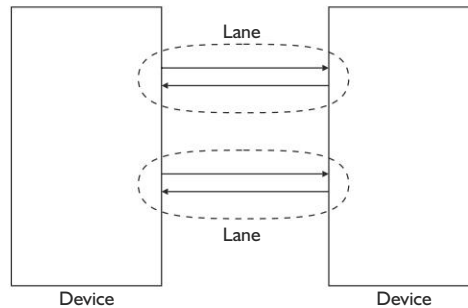
For serial port and parallel port communication, the PCI bus slows down when dealing with low speed devices. Hence, accessing serial and parallel port devices, via a PCI bus, does not cause system timing problems.

10.8.1.7 PCI-Express

PCI provides a 64 bit interface in a 32 bit package. The bus, when running at 33 MHz, can transfer 32 bits of data every clock, with a period of 33 nanoseconds, but memory has a speed of 70 nanoseconds only. When the CPU fetches data from RAM, it has to wait at least three clocks for the data. With faster hard drives, PCI-based sound cards, Ethernet controllers, etc., the PCI bus is unable to meet the speed due to limited bandwidth. Current PCI bus being a multi-drop, parallel bus implementation, has performance limits.

PCI-Express adds a switch replacing the multi-drop bus and distributes I/O messages on a peer-to-peer basis. If one device wants to send data to another, it need not go through the chipset (even though the switch may be part of the chipset). This reduces the amount of data to be handled by the chipset.

Technically, PCI Express is not a bus. It is a point-to-point connection (Fig.10.116) between two devices, and no other device can share this connection. On a motherboard, all PCI slots are connected to the PCI bus, and share the same data path. But, each PCI Express slot is connected to the motherboard chipset using a dedicated lane (data path), not sharing this lane with other PCI Express slots (Fig.10.117). The PCI Express bus is hot pluggable, i.e., it's possible to install and remove PCI Express boards even when the system is powered on.

**Fig. 10.116** PCI Express $\times 2$ bus

The PCI Express bus has been developed to substitute PCI and AGP buses. It is software compatible to the PCI bus; the PCI I/O drivers and OS can directly support the PCI Express bus without any modification.

The PCI Express bus is a serial path operating in full-duplex mode. Data is transmitted in through two pairs of wires called *lane*, by using the codification system 8b/10b, similar to Fast Ethernet (100BaseT, 100 Mbps) networks. Each lane allows a maximum transfer rate of 250 MB/s in each direction. The PCI Express bus can be formed by combining several lanes to obtain higher performance. PCI Express systems with 1, 2, 4, 8, 16 and 32 lanes are available in the market. For example, the transfer rate of a PCI Express system with 8 lanes (x8) is 2 GB/s (250×8).

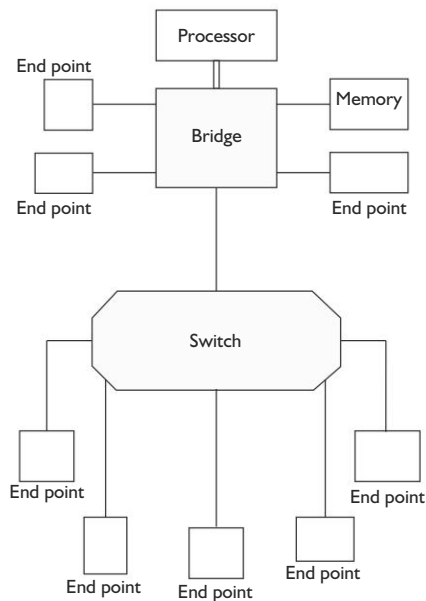
**Fig. 10.117** A system with PCI Express

Table 10.21 compares the transfer rates of the PCI, AGP and PCI Express buses.

TABLE 10.21 Transfer rates of PCI, AGP and PCI Express buses

Sl. no.	Bus	Maximum transfer rate
1	PCI	133 MB/s
2	AGP 2x	533 MB/s
3	AGP 4x	1.066 GB/s
4	AGP 8x	2.133 GB/s
5	PCI Express x1	250 MB/s
6	PCI Express x2	500 MB/s
7	PCI Express x4	1 GB/s
8	PCI Express x16	4 GB/s
9	PCI Express x32	8 GB/s

10.8.1.8 Multiple Buses

The speed of the processor is very high compared to the speed of the memory and I/O devices. To compensate for the mismatch, in PCs, multiple busses are introduced in place of the processor bus as shown in Fig.10.118. The north bridge connects processor to memory and graphic card on AGP at one speed, and to south bridge at another speed, and the south bridge connects I/O devices to north bridge at one speed and PCI slots, at another speed.

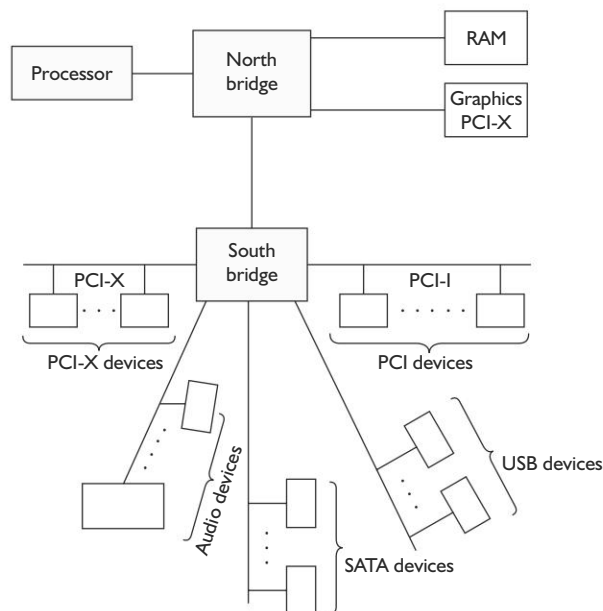


Fig. 10.118 A modern system with multiple I/O buses

10.8.2 SCSI

Small Computer System Interface (SCSI) is an universal parallel I/O interface (Fig. 10.119) for microcomputers, to link multiple peripheral devices of different types on a single I/O bus. It is pronounced as 'Scuzzy'. Each device is given an address on the SCSI bus which is set inside the device by jumpers or DIP switches. The SCSI adapter performs an intelligent protocol sequence on the SCSI bus, to which the connected SCSI devices respond. A 50 pin ribbon cable is used for connecting internal SCSI devices, and thick shielded cable is used for connecting external SCSI devices.

The subsystems on the SCSI bus behave in two different ways: initiator and target. An initiator starts a communication with a target and the target responds to the command from the initiator. The SCSI adapter is an initiator and the peripheral devices are usually the targets. Upto eight subsystems can exist on a SCSI bus. The SCSI uses a handshaking protocol. A device connected to a SCSI bus has to be an intelligent one and is usually costly. Hence SCSI interface is used generally in special servers and RISC systems.

There are three different SCSI standards: SCSI (or SCSI 1), SCSI 2, and SCSI 3.

10.8.2.1 SCSI-1

Its main feature is supporting different types of I/O devices on one interface. This is achieved by isolating the nature of I/O devices from the CPU. Upto eight I/O devices can be attached to a single SCSI interface. The host computer adapter is one of the eight devices. A 3-bit address is used by the SCSI. The eight devices are assigned addresses from 0 to 7. The device with address 7 has the highest priority, and the lowest priority goes to the device with address 0.

The I/O devices are grouped into standard types. The device types currently followed are the following:

- (a) Direct access drives, e.g. hard disk drives
- (b) Printers
- (c) Processors
- (d) WORM devices
- (e) Read-only, direct-access devices

The devices attached to the SCSI interface are intelligent devices with built-in controller logic. The SCSI devices are daisy chained. Each device has two connectors. One for input cable, and the other for output cable. The SCSI bus is terminated on both the sides. On one side, the terminator is present in the host adapter. On the other end, the terminator is present in the last device. There are three types of terminators:

1. Passive termination
2. FPT (Forced Perfect Termination)
3. Active Termination (in SCSI-2)

Passive termination consists of resistors. It can support a maximum of 2 to 3 feet. The FPT consists of clamping diodes for suppressing overshoot and undershoot. The active termination consists of voltage regulator to terminate the interface signals to proper voltage level.

The SCSI supports two types of electrical specifications:

1. Single ended SCSI—This supports upto 6 meters of cable length.
2. Differential SCSI—This supports upto 25 meters cable length. It provides better noise immunity.

In the SCSI, any device can communicate with another in general. But a device may not have the capability of initiating communication. Generally, the host computer is the initiator and the target is one of the device controllers on the SCSI. Depending on the design capability of a device, it can be either initiator always, or target always. A device can have both roles (initiator cum target) also.

In special configuration, more than one host computer can share a common SCSI bus. However, there has to be additional hardware or software to control effective sharing by two computers.

In order to standardise the commands for the SCSI devices, a specification called CCS (Common Command Set) has been developed. It defines a set of commands to be supported by SCSI compatible devices.

10.8.2.2 SCSI-2

The SCSI-2 has been introduced to support new types of devices such as CD-ROMs, scanners, communication devices and optical storage devices (WORM and erasable media). The SCSI-2 supports 8-bit, 16-bit or 32-bit data. The different versions of SCSI-2 are

- (a) Fast SCSI 8-bit, 10 MBPS
- (b) Wide SCSI 16-bit or 32-bit
- (c) Fast and Wide: 16-bit, 20 MBPS or 32-bit, 40 MBPS
- (d) Slow and normal: 8-bit, 5 MBPS

The CCS for SCSI-2 is an enhanced one compared to the CCS for the original SCSI.

10.8.2.3 SCSI-3

The SCSI-3 supports more than eight devices and the transfer speed is 20 MBPS. Its main objective is to support optical fiber and long distance. The wide Fast-20 SCSI gives 40 MBps for 16 bit transfer. The wide Fast-40 SCSI gives 80 MBps for 16 bit devices. Upto 16 devices can be connected on such a SCSI bus.

10.8.2.4 SCSI Bus Principles

The Small Computer System Interface (SCSI) is a parallel I/O bus that supports a variety of peripherals such as disk drives, tape drives, modems, printers, scanners, optical devices, test equipment, and medical instruments. The SCSI provides a peer-to-peer protocol: two devices communicate on equal levels to perform I/O activities. Figure 10.119 shows a typical SCSI configuration.

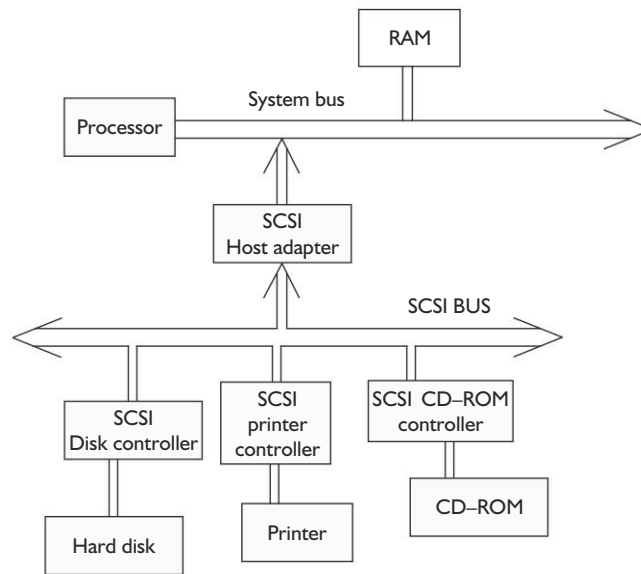


Fig. 10.119 A typical SCSI bus configuration

A maximum of eight SCSI devices (one SCSI host adaptor and seven SCSI device controllers) are supported, though only two devices can communicate at any given time. Each device is assigned a SCSI ID bit: one bit on the Data Bus that corresponds to the device's unique SCSI address (Fig.10.120). An initiator can address up to eight peripheral devices that are connected to a target. These are called Logical Units Numbers (LUN).

The SCSI bus is *time-shared* to achieve effective utilization of the bus bandwidth. A device usually gets disconnected from the bus during internal operation. The devices need the bus to perform data transfer or status transfer. Hence the devices disconnect and reconnect to the bus as and when required. While one device is using the bus, other devices can perform internal operations. When a device is ready to continue transfer, it gets reconnected to the bus. When two devices communicate on the bus, one acts as an initiator and the other as a

target. The initiator initiates an operation and the target performs the operation. A device generally has a fixed role (initiator or target), but some devices can assume either role. Generally, the host is the initiator.

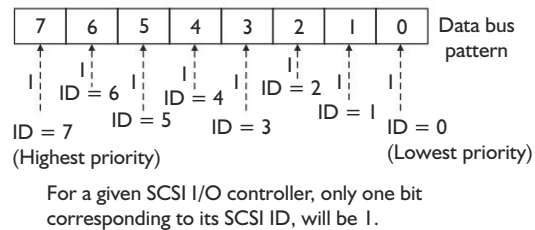


Fig. 10.120 SCSI ID Format

Specific bus roles are defined to the initiators and the targets. The initiator can arbitrate for the bus and select a desired target. The target can request the transfer of Command, Data, Status, or other information, and it can also arbitrate for the bus and reselect an initiator for continuing an earlier specified operation.

Information transfers on the Data Bus are asynchronous and follow a handshake protocol. One byte of information is transferred with each handshake. Synchronous data transfer option is also possible.

10.8.2.5 SCSI Bus Signals

The SCSI bus has eighteen signals: nine control signals and nine data signals including the parity bit. The data signals are used for transfer of messages, commands, status, and data. Table 10.22 describes the SCSI bus signals.

TABLE 10.22 SCSI Bus Signals

Sl. no.	Signal name	Meaning	Type	Description / Meaning	Remarks
1	BSY	Busy	Phase	Indicates that the bus is being used.	An 'OR'-tied signal
2	SEL	Select	Phase	An initiator selects a target, or a target reselects an initiator.	-
3	C/D	Control/Data	Information type	Target conveys whether control or data to be transferred. High for control	Control means command, status or message

(Contd.)

Sl. no.	Signal name	Meaning	Type	Description / Meaning	Remarks
4	I/O	Input/Output	Direction of transfer	Target indicates the direction of data on the data bus.	High during input to the initiator.
5	MSG	Message	Information type	Target identifies the current information as the Message phase.	–
6	REQ	Request	Handshake	Target requests a data transfer cycle	–
7	ACK	Acknowledge	Handshake	Initiator acknowledges data transfer.	–
8	ATN	Attention	Other	Initiator has a message for the target.	–
9	RST	Reset	Other	Initializes all device controllers	An 'OR'-tied signal
10	DB	Data Bus	Data	Eight data-bits signals, plus an odd parity-bit.	Parity unused in Arbitration

10.8.2.6 SCSI Bus Phases

SCSI Bus Phase Sequences

Table 10.23 gives a brief description of different bus phases. The Reset condition can abort any phase and cause the Bus Free Phase. Any other phase can be followed by the Bus Free. The normal sequence is from the Bus Free Phase to Arbitration, from Arbitration to Selection or Reselection, and from Selection or Reselection to one or more of the Information Transfer Phases (Command, Data, Status, or Message). Generally, the final Information transfer Phase is Message In Phase where a Disconnect, Command Complete, or Linked Command Complete message is transferred, followed by Bus Free Phase. The Selection or Reselection phase can occur only after the Arbitration phase. The Arbitration phase can occur only after the Bus Free phase. The Bus Free phase can be entered from any of the other phases. Figure 10.121 shows the normal flow sequence of different phases. Table 10.24 gives status of control signals and their sources in different phases.

TABLE 10.23 SCSI Bus Phases

Sl. no.	Bus phase	Meaning/action
1	Bus free Phase	No device is requesting the bus
2	Arbitration Phase	Any device can arbitrate and win the bus. Devices that lose arbitration can try again when the bus is free
3	Selection Phase	A path is established between the initiator and target. The target controls the bus after the selection phase
4	Information Transfer Phase	Data or control transfer takes place. It is of four different types: Message, Command, Status and Data
5	Message Phase	First information transfer phase in the connection. Initiator sends an Identify message to the target
6	Command Phase	Inquiry command transferred from the initiator to the target
7	Data In Phase	Target responds with Inquiry data
8	Status Phase	Target sends status byte
9	Message In Phase	The last information transferred in the connection is the Command Complete message

TABLE 10.24 Control signals and sources in Bus phases

Bus phase	BSY	SEL	REQ	ATN	DB (P)
BUS FREE	None	None	None	None	None
ARBITRATION	All	Winner	None	None	S ID
SELECTION	I&T	Init	None	I	I
RESELECTION	I&T	T	T	I	T
COMMAND	T	None	T	I	I
DATA IN	T	None	T	I	T
DATA OUT	T	None	T	I	I
STATUS	T	None	T	I	T
MESSAGE IN	T	None	T	I	T
MESSAGE OUT	T	None	T	I	I

Notes:

I – Initiator

T – Target

S ID: A unique data bit (the SCSI ID) activated by the SCSI device that is arbitrating.

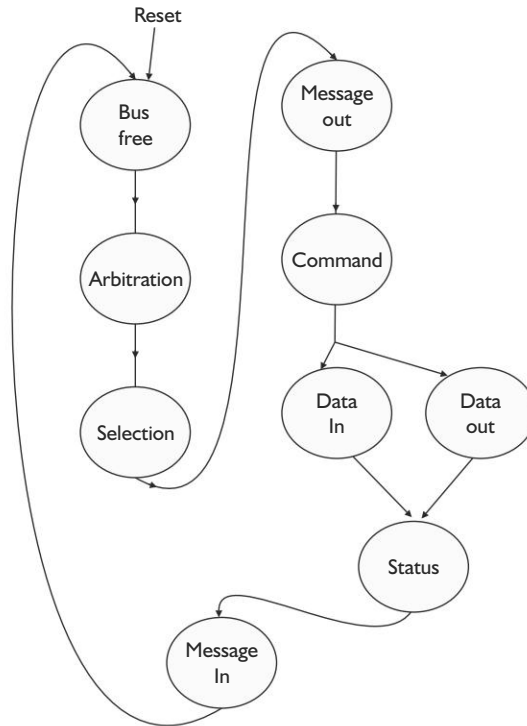


Fig. 10.121 SCSI Bus phase sequences – normal order

Bus Free Phase

In the Bus Free Phase, no device is currently using the SCSI bus.

Arbitration Phase

In the Arbitration Phase, one device gains control of the bus and assumes the role of an initiator or target. The arbitration sequence is as follows:

1. The device waits for the Bus Free Phase: until both BSY and SEL become inactive
2. The device arbitrates for the bus by activating BSY and its own SCSI ID

3. The device checks the Data Bus. (a) If any higher priority SCSI ID bit is active, the device loses the arbitration; it releases its signals and returns to Step 1. (b) If no higher priority SCSI ID bit is active, the device wins the arbitration and it activates SEL. All other devices participating in the Arbitration Phase lose the arbitration
4. The device, having won arbitration, moves to selection phase.

Selection Phase

In the Selection Phase, an initiator selects a target for initiating some function such as Read or Write. (a) The device that won the arbitration has both BSY and SEL activated. It becomes an initiator by deactivating the I/O signal. (b) The initiator sets two SCSI ID bits in the Data Bus: its bit and the target's bit. In fig 10.122, SCSI ID 5 has activated both its own SCSI ID i.e., DB (5), and that of a device i.e., DB (0). (c) The initiator then releases BSY and looks for a response from the target. (d) The target gets selected when SEL and its SCSI ID bit are true, and BSY and I/O are false. The selected target checks the Data Bus, and understands the SCSI ID of the selecting initiator. (e) The selected target then activates BSY. The target does not respond to a selection under following situations: (a) there is parity error. (b) more than two SCSI ID bits are on the Data Bus.

0	1	2	3	4	5	6	7	
1	0	0	0	0	1	0	0	Data bus

Fig. 10.122 Selection phase SCSI IDs

Reselection Phase

By Reselection phase, a target reconnects to an initiator to continue the operation that was earlier suspended by the target. For example, for Read command, the hard disk controller can disconnect if a time consuming seek operation has to be done before read operation. The reselection phase sequence is similar to selection phase but for one difference: the target reselects an initiator instead of an initiator selecting a target.

(a) On completing the Arbitration Phase, the winning device has both BSY and SEL activated. It becomes a target by asserting the I/O signal. (b) The winning device also sets the Data Bus with its SCSI ID bit and the initiator's SCSI ID bit. (c) The target then releases BSY and waits for a response from the initiator. (d) The initiator gets reselected when SEL, I/O and its SCSI ID bit are true and BSY is false. The reselected initiator checks the Data Bus and understands the SCSI ID of the reselecting target. (e) The reselected initiator then

activates BSY. The initiator does not respond to a Reselection Phase in case of parity error or if other than two SCSI ID bits are on the Data Bus. (f) After the target detects BSY, it also activates BSY and releases SEL. (g) The target may then change the I/O signal and the Data Bus. (h) After the reselected initiator detects SEL false, it releases BSY. The target continues issuing BSY until it releases the bus.

Information Transfer Phase

Information Transfer Phase involves three types of communication: (a) Command from initiator to target (b) Status from target to initiator (c) Data to or from the device. The C/D, I/O, and MSG signals identify the exact nature of Information Transfer. The target activates these signals, and controls change of phase from one to another. The initiator can request a Message Out Phase by activating ATN, whereas the target can cause the Bus Free Phase by releasing MSG, C/D, I/O and BSY. This Phase uses REQ/ACK handshakes to control the information transfer. For each handshake, one byte of information is transferred. During this phase, BSY remains active and SEL remains inactive.

In the Command Phase, the target requests command from the initiator. The target activates the C/D and negates the I/O and MSG during the REQ/ACK handshake. The Data Phase is of two types: Data In Phase and Data Out Phase. In the Data In Phase, data has to be sent to the initiator from the target. The target activates the I/O signal and negates the C/D and MSG signals during the REQ/ACK handshake. In the Data Out Phase, data has to be sent from the initiator to the target. The target negates the C/D, I/O, and MSG signals during the REQ/ACK handshake. In the Status Phase, status has to be sent from the target to the initiator. The target asserts C/D and I/O and negates the MSG signal during the REQ/ACK handshake.

Asynchronous Information Transfer

The target controls the direction of information transfer by the I/O signal. When I/O is HIGH, transfer is from the target. When I/O is LOW, transfer is from the initiator.

(a) If I/O is HIGH, the target places information on DB and activates REQ. The initiator reads DB, and conveys acceptance by activating ACK. The target can place a new data or release DB and negate REQ. The initiator then negates ACK. (b) If I/O is LOW, the target requests information by activating REQ. The initiator places data on DB and activates ACK. The target reads DB, and negates REQ. The initiator can modify or release DB, and negate ACK.

Synchronous Data Transfer

Synchronous data transfer is used in data phases provided a synchronous data transfer *agreement* has been already established between the target and initiator. The REQ pulses can be sent by the target in advance of the number of ACK pulses received from the initiator, establishing a pacing mechanism.

Message Phase

The Message Phase is of two types: Message In Phase and Message Out Phase. In the Message In Phase, message has to be sent to the initiator from the target. The target asserts C/D, I/O, and MSG during the REQ/ACK handshake. In the Message Out Phase, message has to be sent from the initiator to the target. The target invokes this phase in response to the Attention condition created by the initiator. The target asserts C/D and MSG and negates I/O during the REQ/ACK handshake. The target handshakes byte(s) until ATN is negated, except when rejecting a message.

If the target detects parity error on the message byte(s), it requests for retry by asserting REQ after detecting ATN has gone false (prior to changing to any other phase). The initiator sends the previous message byte(s). If the target receives the message byte(s) successfully, it changes to any information transfer phase other than the Message Out Phase and transfer at least one byte.

SCSI Bus Conditions

The SCSI bus has two asynchronous conditions: Attention and Reset. These conditions cause the device to perform certain actions and can alter the phase sequence. By the Attention condition, an initiator informs that it has a message ready. The target can get this message by performing a Message Out Phase. The initiator creates the Attention condition by activating ATN. The Reset condition clears all devices from the bus. Any SCSI device may create the Reset condition by activating RST. All SCSI devices release bus signals (except RST). The Bus Free Phase always follows the Reset condition.

Hard Reset

On detecting the Reset condition, the device does following actions:

1. Clears all I/O processes
2. Releases all device reservations
3. Returns any SCSI device operating modes to their appropriate initial conditions
4. Unit Attention condition is set.

10.8.2.7 Messages

The first message sent by the initiator after the Selection Phase is the Identify, Abort, or Bus Device Reset message. The Identify message establishes a logical connection between the initiator and the specified logical unit. After the Reselection Phase, the target's first message will be Identify. Table 10.25 lists few typical messages.

TABLE 10.25 Typical Messages

Sl. no.	Message	Source and destination	Description
1	Abort	From the initiator to the target	Clears the present I/O process. The target enters the Bus Free Phase.
2	Bus Device Reset	From initiator to target	Clears all current I/O processes; forces a hard reset.
3	Command Complete	From target to initiator	Execution of a command is terminated; status sent to the initiator
4	Disconnect	From either way	The target informs that the present connection will be broken. OR The initiator instructs the target to disconnect.
5	Identify	From either way	Forms I - T - L (or I-T-R) nexus. Addresses a logical device attached to a target.
6	Initiator Detected Error	From initiator to target	An error has occurred.
7	Message Parity Error	From initiator to target	Last message byte has a parity error
8	Message Reject	From either way	Last message byte is inappropriate or not been implemented.
9	No Operation	From initiator to target	Presently, initiator does not have any other message.

10.8.2.8 Command Descriptor Block

A request to a peripheral device is performed by sending a command descriptor block to the target. The request is accompanied by parameters sent during the Data Out Phase. The command descriptor block consists of following information: operation code, logical unit number, command parameters, and a control byte.

10.8.2.9 Status

The status byte is sent from the target to the initiator during the Status phase. Table 10.26 lists typical status codes. Fig. 10.123 shows a typical SCSI Bus sequence and timing.

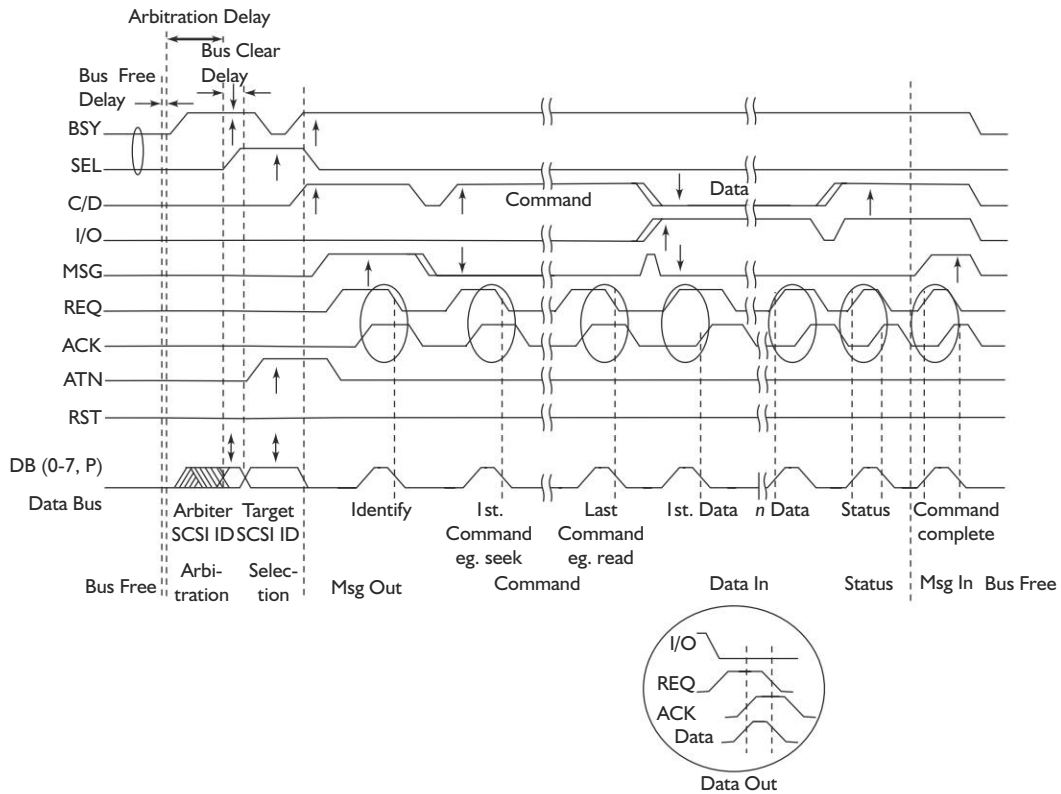


Fig. 10.123 SCSI bus sequence

TABLE 10.26 Status Definitions

Sl. no.	Status code	Description	Remarks / Further action required
1	Good	Target has successfully completed the command	–
2	Check Condition	An error, exception, or abnormal condition has Occurred	Request Sense command to be issued to determine the nature of the condition
3	Busy	Target is busy	Initiator to issue the command again later.
4	Reservation	Access to a reserved unit	Issue the command again later

10.8.3 USB

Universal Serial Bus (USB) is a serial I/O interface between peripheral devices and a computer. Since it enables easy connection of peripherals to a computer, it has replaced the RS232 interface and old parallel interfaces. USB has become a successful I/O interface in personal computing, consumer electronics and mobile products: it connects devices like mouse, keyboard, PDA, game-pad and joystick, scanner, digital camera, printer, personal media player, flash drive and hard disk drive, to computers. The USB was developed jointly by many leading companies like Intel, Compaq, Microsoft, Digital, IBM etc. The USB Implementers Forum (USB-IF) is a non-profit corporation that facilitates the development of USB compatible peripherals. USB initially supported maximum speed of 12 Mbps. Present versions support four different speeds: (a) A low speed of 1.5 Mbps for low-bandwidth devices such as keyboard, mouse, and joystick (b) The full speed of 12 Mbps (c) A hi-speed of 480 Mbps (d) A SuperSpeed of 4.8 Gbps.

10.8.3.1 USB Advantages

The USB is preferred by everyone due to benefits such as low cost, high performance, expandability, auto-configuration and hot-plugging. The USB offers several attractive features as listed below:

1. Simple connectivity; provides a single type of connector for all types of devices.
2. No need for switch setting, for peripheral addresses and interrupt levels.
3. Supports variety of data from slow mouse inputs to digitized audio and compressed video.
4. Permits hot swapping; the devices can be plugged or unplugged without powering off the system.
5. Allows “plug and play”; when a device is connected to USB bus, it is recognized by the host computer and configured, without any effort by the user.
6. Support for a large number of devices; up to 127 devices can be used on USB port.
7. It supplies power, on the bus, eliminating need for a separate power supply in peripherals.
8. The OTG feature allows a portable device such as cell phones and digital cameras, when connected as a USB peripheral, to directly connect to other USB devices. A user can send photos from a digital camera to other peripherals such as printer, PDA, cell phone etc., or send music files from an MP3 player to another portable player, PDA or cell phone.

10.8.3.2 USB Versions

USB1.0: USB 1.0 allows transfer at 12 Mbps. Known as full-speed USB, it supports a wide range of devices, including MPEG video devices and digitizers.

USB1.1: USB version 1.1 supports two modes: full speed mode of 12Mbps and low speed mode of 1.5Mbps. The 1.5Mbps mode is less prone to EMI.

USB2.0: USB 2.0 supports three modes: 1.5, 12 and 480 Mbps. It supports both low-bandwidth devices such as keyboard and mouse, and high-bandwidth devices such as high-resolution Web-camera, scanner and printer. Known as hi-speed USB, it supports a transfer rate of up to 480 Mbps. The USB On-The-Go (OTG), a supplement to USB 2.0, supports dual-role devices that can function either as hosts or as peripherals, so that two USB devices can communicate with each other directly.

USB3.0: Known as Super-Speed USB, USB 3.0 supports data transfer rates of 4.8 Gbps. USB 3.0 products are expected to be available in 2010, to support high-density digital content and media.

Wireless USB: The USB wireless networking standard will use ultra-wideband wireless technology for data rates of up to 480 Mbps.

10.8.3.3 USB Principles of Operation

The host, being the USB master, manages all communications and the peripheral devices respond to commands received from the host. The USB uses a seven-bit address. Hence it can support up to 127 different devices, excluding the “all zeroes” code. A physical USB device can consist of multiple logical devices that are identified as device functions. Each logical device is allotted a unique address by the host. For example, a webcam with a built-in microphone, has two functions each having a distinct address.

Device Classes

USB classifies the devices into different classes and assign class codes. The class code for any device identify the device’s functionality, so that appropriate device driver can be used. Some typical examples for device classes are listed in Table 10.27.

TABLE 10.27 USB Typical device classes

Class (Hexadecimal)	Function	Description	Examples
01	Interface	Audio	Speaker, microphone, sound card
02	Both interface and device	Communications and CDC Control	Ethernet adapter, modem, serial port adapter
03	Interface	Human Interface Device (HID)	Keyboard, mouse, joystick
05	Interface	Physical Interface Device (PID)	Force feedback joystick
06	Interface	Image	Webcam, scanner
07	Interface	Printer	Laser printer, inkjet printer, CNC machine
08	Interface	Mass Storage	USB flash drive, memory card reader, digital audio player, digital camera, external drive
09	Device	USB hub	Full speed hub, hi-speed hub
0A	Interface	CDC-Data	This class and class 02 (Communications and CDC Control) are used together.
0B	Interface	Smart Card	USB smart card reader
0E	Interface	Video	Webcam
E0	Interface	Wireless Controller	Wi-Fi adapter, Bluetooth adapter

USB System Topology

The USB system consists of a host, USB hubs and peripheral devices connected in a tiered-star topology as shown in Fig. 10.124. USB devices are connected through hubs. The host controller and the root hub are part of the computer system's hardware. A USB host computer can have multiple host controllers and each host controller can provide multiple USB ports. On a USB hub, the port used to connect to the host either directly, or via another hub is known as the upstream port, and the ports used for connecting other devices to the USB hub are known as the downstream ports. Each USB hub routes data from the host to its correct destination. The host controller manages traffic flow to devices. In USB 2.0, the host controller polls the bus in a round-robin fashion. USB device can not transfer data on the bus without a request from the host controller. In USB 3.0, devices can request service from host. In USB, users can connect or remove peripherals, without powering off the system.

Hubs detect these topology changes. They also source power to the USB network. The power can come from the hub (if it has a built-in power supply), or can be passed through, from an upstream hub.

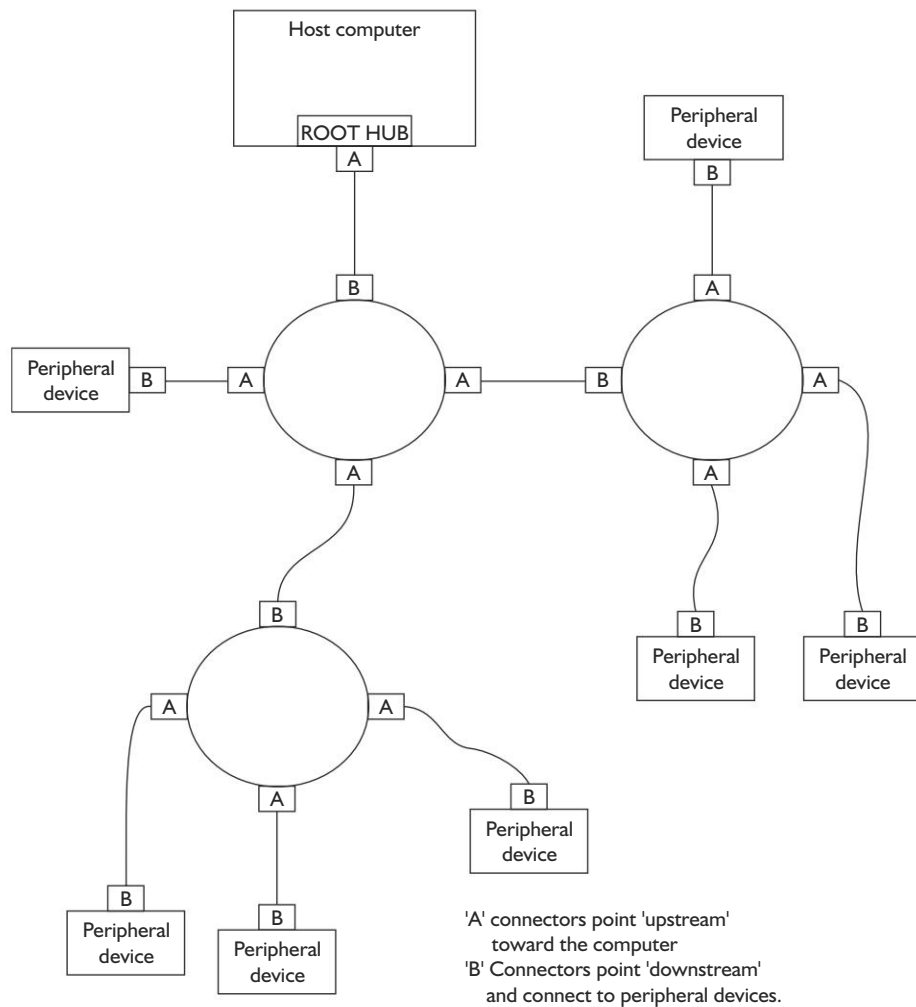


Fig. 10.124 USB 'tiered star' topology

USB Cable

The USB cable has an “A” connector on one end and a “B” on the other. “A” connectors head “upstream” toward the computer, and “B” connectors head “downstream” and con-

nect to individual devices. Length of USB cable can be up to 5 meters for 12Mbps connections and 3 meters for 1.5Mbps. Using hubs, devices can be up to 30 meters away from the host. The USB uses differential transmission for data on a twisted pair of wires. Another pair is used for power to downstream peripherals; +5 volts and ground. Low-power devices such as mouse can draw power from the bus. High-power devices such as printer have their own power supplies. Hubs can have their own power supplies, and also provide power to devices. USB hosts and hubs manage power by enabling and disabling power to individual devices. They can also command devices to enter the suspend state, in order to reduce power consumption.

Enumeration

When a USB device is first connected to a USB host, the USB device enumeration process is started so that the host can learn the identity of the device, and decide, which device driver is required. The enumeration process, first sends a reset signal to the USB device. The speed of the USB device is determined during the reset sequence. After reset, the device's information is read by the host, and then the device is assigned a unique 7-bit address. If the device is supported by the host, the relevant device driver is loaded and the device is set to a configured state. Whenever a hub detects a new peripheral (or removal of a peripheral), it reports this information to the host and the enumeration process starts. When the USB host is restarted also, the enumeration process is done for all devices.

USB Communication

The USB communications take place between the host and endpoints. An endpoint is a uniquely addressable part of the peripheral that functions as the source or receiver of data. Four bits are used for the device's endpoint address; codes also indicate transfer direction, and whether the transaction is a control transfer. Endpoint 0 is reserved for control transfers, and up to 15 bi-directional endpoints can exist within each device. All devices have endpoint zero which receives device control, and status requests during enumeration and throughout the operation.

USB transfers occur through virtual pipes that connect the peripheral's endpoints with the host. When establishing communications with the peripheral, each endpoint sends a *descriptor* that indicates the endpoint's configuration and expectations. The various details include transfer type, maximum size of data packets, the time interval for data transfers, and the bandwidth needed. With these details, the host establishes connections to the endpoints through virtual pipes. A USB device can have 16 pipes coming into the host controller and

16 going out of the controller. The pipes are unidirectional. Endpoints are grouped into interfaces and each interface is associated with a single device function. For easy visualization, the reader can compare the end point with an I/O port inside an I/O controller such as a hard disk controller or floppy disk controller. Each of these controllers has multiple I/O ports, each for a distinct role.

USB supports four data transfer types: Control, Isochronous, Bulk and Interrupt. There are two types of pipes: stream and message pipes. A stream pipe is a uni-directional pipe connected to a uni-directional endpoint and used for bulk, interrupt, and isochronous data flow. A message pipe is a bi-directional pipe connected to a bi-directional endpoint and used for control data flow.

Control transfers involve transfer of configuration, setup and command information between the device and the host. The host can send commands or query parameters with control packets. Isochronous transfer is used by time-critical, streaming devices such as video cameras. Data is transferred between the device and the host in real-time at regular intervals. It should get access to the USB bus without undue delay. Bulk transfer is used by devices such as printers and scanners, which receive data in one big packet. For these, the timely delivery is not critical. Hence the bulk transfers are fillers, utilizing unused USB bandwidth. Interrupt transfers are used by peripherals that need immediate attention. It is used by devices to request some service from the host. Devices such as mouse and keyboard belong to this category.

The USB divides the available bandwidth into frames, and the host controls the frames. Frames contain 1,500 bytes, and a new frame starts every millisecond. During a frame, isochronous and interrupt devices get a slot so they are guaranteed the bandwidth they need. Bulk and control transfers use whatever space is left. As devices are enumerated, the host keeps track of the total bandwidth requested by all the isochronous and interrupt devices. They can consume up to 90 percent of the total available bandwidth. After this, the host denies access to any other isochronous or interrupt devices. Control packets and packets for bulk transfers use the remaining bandwidth of 10 percent.

USB Packets and Formats

Data transfer is in the form of packets transfer between the host and peripheral devices. Initially, all packets are sent from the host, via the root hub and other hubs, to devices. Some of those packets direct a device to send some packets in reply. A USB connection is between a host or hub at the “A” connector end, and a device or hub’s “upstream” port at the other end. A USB packet begins with an 8-bit synchronization sequence 00000001.

USB Packets consist of the Following Fields:

1. **Sync field:** The sync field synchronizes the clock of the receiver with that of the transmitter. The sync field is 8 bits long at low and full speed, or 32 bits long for high speed. The last two bits indicate where the PID fields starts.
2. **PID field:** This field (Packet ID) identifies the type of packet. It consists of the 4-bit PID followed by its bit-wise complement, making an 8-bit PID. This redundancy helps error detection.
3. **ADDR field:** This field specifies the destination device for which the packet is designated for.
4. **ENDP field:** This field gives 4 bits, allowing 16 possible endpoints.
5. **CRC field:** Cyclic Redundancy Checks are for error detection for the data within the packet payload. Token packets have a 5-bit CRC while data packets have a 16-bit CRC.
6. **EOP field:** This indicates End of packet.

The USB Packets are of four Basic Types: Handshake packet, Token packet, Data packet, PRE packet and Start of Frame Packet. As already mentioned, the host initiates all transactions. The first packet is a token that indicates the following: (a) what follows (b) whether the data transfer is a read or write (c) device address and (d) designated endpoint. The next packet is generally a data packet containing the content information. It is followed by a handshaking packet, reporting if the data or token was received successfully, or if the endpoint is stalled, or not available to accept data.

Handshake Packet

Handshake packet (Fig.10.125) is generally sent in response to data packet. It consists of a PID byte. The three basic types of handshake packets are ACK, NAK and STALL. USB 2.0 added two additional handshake packets: NYET and ERR. Table 10.28 defines the function of the five types of handshake packet. The only handshake packet, the USB host generates is ACK. If the host is not ready to receive data, it does not instruct a device to send any.



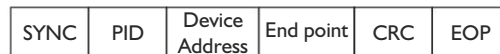
Fig. 10.125 Handshake packet

TABLE 10.28 Handshake Packets

Sl. no	Handshake Packet type	Meaning
1	ACK	Data packet successfully received.
2	NAK	Data packet not received, and should be retransmitted.
3	STALL	Error in device; corrective action is required.
4	NYET	A split transaction is not yet complete.
5	ERR	A split transaction failed.

Token Packet

Token packet (Fig. 10.126) consists of a PID byte followed by 11 bits of address and a 5-bit CRC. Tokens are sent by the host, not by a device. There are three types of token packets: In token, Out token and Setup token. USB 2.0 added PING and SPLIT token. Table 10.29 defines the five types of token packets.

**Fig. 10.126** Token packet**TABLE 10.29** Token packets

Sl. no.	Token Packet type	Meaning / Function
1	In token	Informs the USB device that the host wishes to read information
2	Out token	Informs the USB device that the host wishes to send information
3	Setup token	Used to begin control transfers.
4	PING token.	Asks if device is ready to receive.
5	SPLIT token	Similar to OUT token; used for initial device setup.

An endpoint of a pipe is addressable with device address, endpoint number as specified in a TOKEN packet. If the direction of the data transfer is from the host to the endpoint, an OUT packet with the desired device address and endpoint number is sent by the host. If the direction of the data transfer is from the device to the host, the host sends an IN packet. A bi-directional endpoint accepts both IN and OUT packets.

IN and OUT tokens contain a 7-bit device number and 4-bit function number and command the device to transmit DATA-packets, or receive the following DATA-packets, respectively. An IN token expects a response from a device. The response may be a NAK or STALL response, or a DATA frame. If the response is a DATA frame, the host issues an ACK handshake if appropriate. An OUT token is followed immediately by a DATA frame. The device responds with ACK, NAK, or STALL depending on the situation. The SPLIT token has a 7-bit hub number, 12 bits of control flags, and a 5-bit CRC. This is used to perform split transactions when communicating with slow speed devices. Instead of keeping the high-speed USB bus busy in sending data to a slower USB device, the nearest high-speed capable hub is given a SPLIT token followed by one or two USB packets at high speed. It then performs the data transfer at full or low speed, and provides the response at high speed, when prompted by a second SPLIT token.

SETUP token is used for initial device setup. It is followed by an 8-byte DATA0 frame with a standardized format. The PING token, asks a device if it is ready to receive an OUT/DATA packet pair. The device responds with ACK, NAK, or STALL, as appropriate.

Data Packet

A data packet (Fig.10.127) consists of the PID followed by 0–1023 bytes of data payload (up to 1024 in high speed, maximum 8 at low speed), and a 16-bit CRC. There are two basic data packets DATA0 and DATA1. Both consist of a DATA PID field, 0-1023 bytes of data payload and a 16-bit CRC. They must be preceded by an address token. The data packets are usually followed by a handshake token, as response from the receiver. USB 2.0 includes DATA2 and MDATA packet types. They are used only by high-speed devices for isochronous transfers.



Fig. 10.127 Data packet

Since the alternate DATA packets are numbered as 0 and 1, the receiver can monitor if any DATA packet is lost. Similarly the sender can monitor if any ACK is lost. This is similar to the Stop-and-wait ARQ protocol used in networks. The sender should receive the receiver's response within a timeout period. If a USB host does not receive a response (such as an ACK) for data packet, there are two possibilities: (a) the data is not received (Fig. 10.128) by the device (b) data has been received (Fig.10.129) but the handshake response by the receiver has been lost. In both cases, the host will resend the original DATA packet. To

differentiate these two cases, the device keeps track of the type (0 or 1) of DATA packet it last accepted. If it receives another DATA packet of the same type, it is acknowledged, but ignored, since it is a repetition. Only a DATA packet of the opposite type is actually accepted.

When a device is reset with a SETUP packet, it expects an 8-byte DATA0 packet next. Table 10.30 defines different types of data packets.

TABLE 10.30 Data packets

Type	Name	Description
Data	DATA0	Even-numbered data packet
	DATA1	Odd-numbered data packet
	DATA2	Data packet for high-speed isochronous transfer (USB 2.0)
	MDATA	Data packet for high-speed isochronous transfer (USB 2.0)

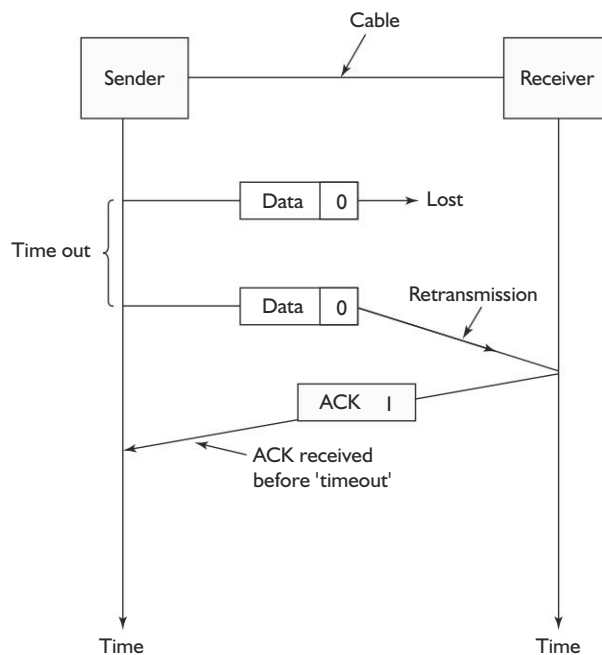
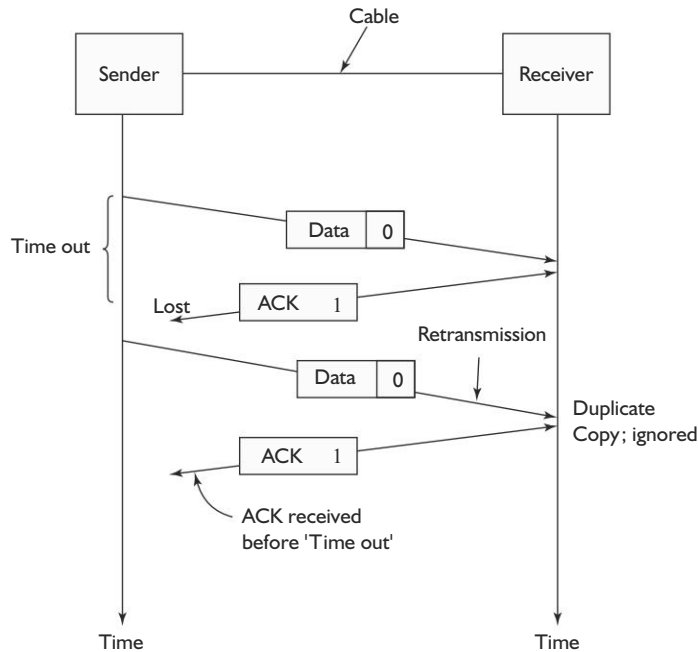


Fig. 10.128 Lost data frame

**Fig. 10.129** Lost ACK frame**Start of Frame Packet**

The USB host transmits a special SOF (start of frame) token (Fig.10.130), every 1 ms. It contains an 11-bit incrementing frame number. This synchronizes isochronous data flows. USB 2.0 devices receive 7 additional duplicate SOF tokens per frame.

**Fig. 10.130** Start of frame packet

Notes: PID – Packet ID; CRC – Cyclic Redundancy Check; EOP – End of Packet

PRE Packet

PRE packet is a preamble meant for low-speed devices and marks the beginning of a low-speed packet. It is used by hubs, which normally do not send full-speed packets to low-speed devices. Full-speed devices other than hubs ignore the PRE packet and its low-speed contents, until the final SE0 indicates that a new packet follows.

USB 3.0

In USB 3.0, full-duplex mode is used when operating in SuperSpeed mode. SuperSpeed establishes a communications pipe between the host and each device. USB 3.0 extends the bulk transfer type in SuperSpeed with Streams. This extension allows a host and device to transfer multiple streams of data through a single bulk pipe.

USB Vs FireWire

1. USB is simple and cheaper, whereas FireWire offers higher performance.
2. USB uses a tiered-star topology, while FireWire uses a tree topology.
3. In USB 1.0 and 2.0, devices cannot initiate communication with the host until the host specifically requests such communication. USB 3.0 allows device-initiated communications. A FireWire device can initiate communication with host.
4. In a USB system, a single host controls the network. In a FireWire network, any node can control the network.

SUMMARY

A peripheral device is interfaced to the system nucleus using a device controller. The device controller is connected to the system interface (bus) on one side and to the device interface on the other side. For a given system, the system interface for all I/O controllers is the same whereas the device interface depends on the device type. Some examples of device interface are: RS-232C, Centronics Interface, SA450, ST-506, SCSI, IDE, USB, FireWire. There are two types of device interfaces: Serial and Parallel. In *Serial interface*, the data bits are released one after the other serially on a single line whereas in *Parallel interface* eight data bits (one byte) are transmitted simultaneously, on different lines.

The I/O drivers perform various I/O operations by issuing a sequence of commands to the peripheral controllers. The controller generates appropriate control signals depending on the command received from the CPU/device driver. Each status signal reports a specific status of the device.

There are four popular methods of designing an I/O controller hardware:

- (a) Hardwired design (Random Logic)
- (b) Special purpose LSI based design
- (c) Microprocessor based design
- (d) Custom built ICs or gate arrays

An I/O port is a program addressable hardware unit with which the CPU can transfer information. In a direct I/O scheme, the I/O ports are treated differently from the memory locations and the program uses IN or OUT instruction for communication. A program reads the data from an input port by using an IN instruction. The OUT instruction transfers data from the processor register to the output port. In a memory mapped I/O scheme, the I/O ports are treated as memory locations. Hence, IN or OUT instructions are not used.

A bus is a group of signals common to several hardware units. These signals are of three types:

- (a) Address bus signals
- (b) Data bus signals
- (c) Control bus signals

The bus cycle is a sequence of events for transferring one word of information.

When two units communicate with each other, one of them is the master and the other becomes slave. In the synchronous method of transfer, the sending and receiving units are supplied same clock. Each action is synchronized either to the raising edge or falling edge of the clock. In the asynchronous transfer, there is no common clock and the master and slave can follow some sort of acknowledgment protocol during data transfer sequence. Interrupt is a signal or event in a computer system, due to which the CPU temporarily suspends the current program and executes another program known as Interrupt Service Routine (ISR). Interrupt Handling is performed jointly by hardware and software. There are three actions performed automatically by the CPU on sensing an interrupt:

- (a) Saving CPU status
- (b) Finding out the interrupt cause
- (c) Branching to the ISR

The CPU can be instructed to disable the interrupts by using the IE flag. While executing an interrupt service routine, if the CPU allows another interrupt, it is known as interrupt nesting. The interrupt controller assigns a fixed priority for the various interrupt requests. If the program intends to allow only certain interrupts and disable others, it can be done by selective masking. The mask pattern, given by the program, decides which interrupts are masked at a time. Non-Maskable Interrupt (NMI) is raised for urgent attention of the CPU (ISR). The processor attends to NMI immediately. The sensing of NMI does not depend on EI flag.

The term 'Data Transfer' refers to the exchange of information (program and data) between the CPU/memory and I/O devices. The data transfer is carried out usually in steps of one word, at a time. The software method of data transfer is of two types:

- (a) Programmed mode (Polling)
- (b) Interrupt mode

The *programmed mode* of data transfer is a dedicated task performed at a stretch (from the beginning of the first byte to the end of the last byte) by the software. In the *interrupt mode*, the software (CPU) is invoked by the hardware only during the actual data transfer of every byte and at other times it is free to do any other task. The DMA mode is a hardware method. The DMA controller or data channel takes care of transferring data between memory and I/O device. The software issues relevant information about the I/O operation to the DMA controller. It has two distinct advantages over the programmed mode and interrupt mode:

- (a) Supporting high speed peripheral devices
- (b) Achieving parallelism between CPU processing and I/O operation

High performance and flexibility are the attractive features of the modern serial interfaces such as Universal Serial Bus (USB) and FireWire. They also support a wide range of peripheral devices—from keyboard to digital camera.

Bus arbitration is a process of determining the bus master (who would have the bus control) when there is a request for bus mastership from many bus masters. The ‘bus arbiter’ decides who should become the current active bus master. The others should wait for their turn. In most computers, the processor and DMA controller are bus masters whereas memory and I/O controllers are slaves. In some systems, intelligent I/O controllers also act as bus masters. The communication in a modern computer system takes place over different types of bus: Local bus, System bus, I/O buses and Mezzanine bus.

REVIEW QUESTIONS

1. In any modern computer system, the application program is not allowed to perform I/O operations without the knowledge of the operating system in order to prevent unauthorized access to the files. In spite of this, a program can cause system ‘hang’ during an I/O operation by mischief or malfunction. Suggest a solution.
2. A program executes 10,000 instructions. It alternately reads data from the floppy and writes data on the floppy after every 1000 instructions. The user carelessly removes the diskette when the program has completed 3100 instructions and inserts a different diskette. If this mistake is unchecked, it will result in reading wrong data and overwriting the second floppy. Hence, the operating system should detect this abnormal situation. Suggest a hardware support (present in the controller/device/interface) that can be used to alert the operating system.
3. A processor development team always supports only the memory mapped I/O and never provides direct I/O for achieving cost reduction. Explain how the processor cost is increased by including support for direct I/O (isolated I/O).

4. The DMA facility allows parallelism between CPU and I/O transfer with a limitation: the CPU cannot use the bus if an I/O transfer is in progress. As an improvement, a designer proposed dual port memory connected on two different buses: one for communication with CPU and the other for I/O transfer. Though this provides full parallelism, the hardware cost increases due to additional circuits. Another designer proposed having I/O memory as a separate module, physically present in the I/O controller, but logically in the main memory space (equivalent to the video buffer in the CRT controller). What are the merits and demerits of the second approach?
5. Synchronous communication does not add start/stop bits to each character unlike asynchronous communication. Still, it is not preferred in low speed data transfer. Why?

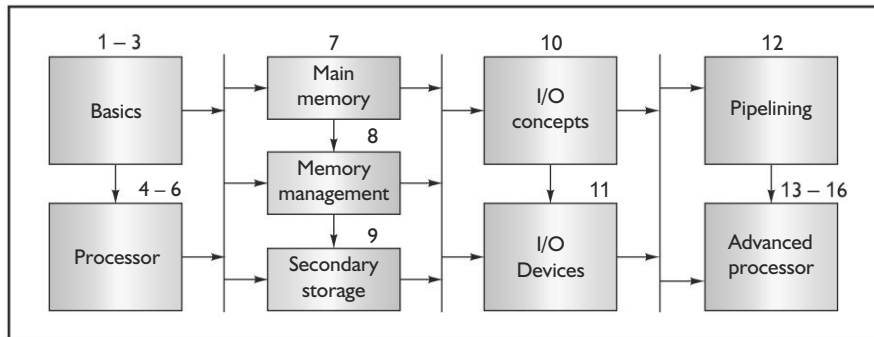
EXERCISES

1. A microcomputer based on Intel 80286 microprocessor operates at zero-wait state at a clock frequency of 12 MHz. Calculate the maximum bandwidth offered by the system. Each bus cycle transfers two bytes and there are two states (clock periods) in one bus cycle since we have a zero-wait state system.
2. A 64 bit PCI system operates at 66 MHz. Calculate the bandwidth.

CHAPTER 11



I/O DEVICES



Chapter Flow Diagram

OBJECTIVES

- « Keyboard
- « CRT Display
- « Printers
- « Mouse
- « Modem
- « Scanner
- « Digital camera
- « Touch pad
- « Flat panel display
- « Graphics accelerator
- « Cable modem

KEYWORDS

Keyswitch
Scanning
Electron Beam
Deflection
HSYNC
VSYNC
TTL Interface
Composite Video
Character Box
Video Bandwidth
Impact Printer
Character Printer
Line Printer
Carriage Motor
Print Head

Paper Feed
Inkjet Printer
Fax-Modem
Mouse
Laser Printer
Flat Panel Display
VRAM
Splitter
RAMDAC
Graphic Tablet
Light Pen
Joystick
Trackball
Zip Drive
LED Printer

 11.1 Introduction

Peripheral devices are the agents through which one interacts with a computer. Earlier, the peripheral devices were huge electromechanical machines with several circuits and mechanisms. The present day peripherals are small and powerful. This chapter presents the overall organization of popular peripherals used in a common configuration: Keyboard, CRT Monitor, Printer, Mouse, Modem and Scanner. Floppy disk drive, Hard disk drive and CD-ROM drive have been already covered in Chapter 9, as secondary storage devices.

Apart from these common peripherals, there are some special peripherals used in certain applications: Digital camera, Digitiser, Joystick, Light pen, Touch Pad and Plotter. Some of these are briefly discussed in this chapter. As application of computers is spreading rapidly, new types of peripheral devices are being developed to satisfy the functional requirements. In addition, performance improvements in existing peripheral devices are achieved frequently.

 11.2 Keyboard

The keyboard is the most friendly input peripheral. Both program and data can be keyed in through it. In addition, certain commands to software can be given from the keyboard.

The keyboard consists of a set of keyswitches. There is one keyswitch for each letter, number, symbol etc., much like a typewriter. When a key is pressed, the keyswitch is activated. The keyboard has an electronic circuit to determine which key has been pressed. Then, a standard 8 bit code is generated, and sent to the computer. Detecting which key is pressed, and generating the corresponding code, is known as *encoding*.

A serial keyboard sends the data, bit-by-bit, in a serial fashion, as shown in Fig. 11.1. The computer converts the data into a parallel byte. The wireless keyboard is a recent development, which does not need physical cable and allows convenient location of the keyboard.

11.2.1 Keyboard Function

Figure 11.2 shows the block diagram of a keyboard. Generally, the keyswitches are connected in a matrix of rows and columns. The functions to be performed by the keyboard electronics are:

1. Sensing a key depression
2. Encoding
3. Sending the code to computer.

There are different types of keyswitches. Some of the common types are:

1. Mechanical keyswitch
2. Membrane keyswitch
3. Capacitive keyswitch

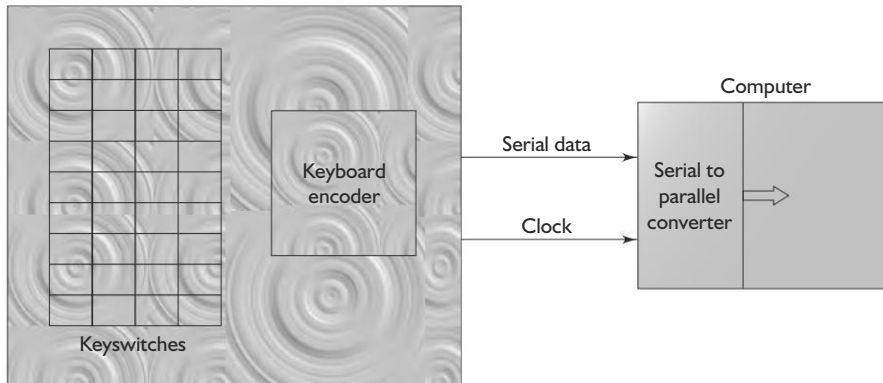
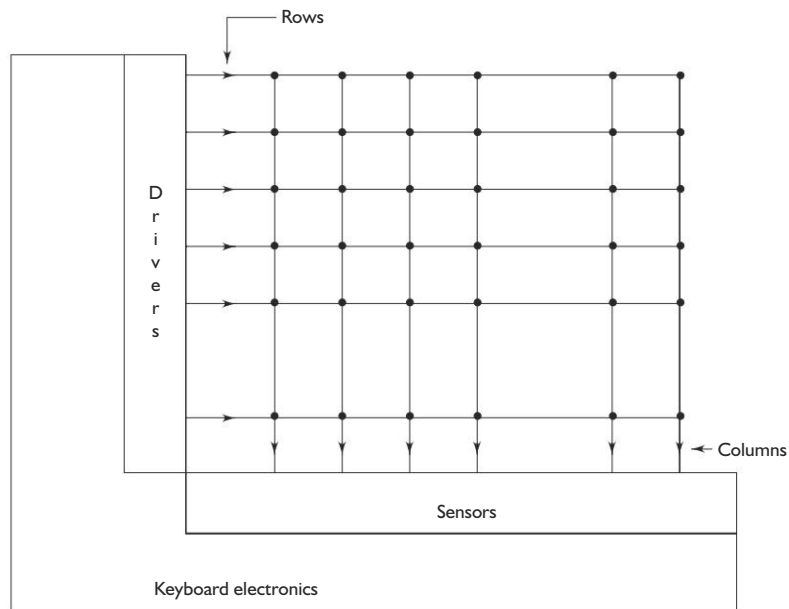


Fig. 11.1 Serial keyboard



Note: Each dot represents a switch.

Fig. 11.2 Keyboard block diagram

11.2.1.1 Mechanical Keyswitch Keyboard

Figure 11.3(a) shows the construction of a mechanical keyswitch. In this type of keyboard, the key is fixed on a plunger. On pressing the key, the plunger moves down making a connection between two points in the keyboard matrix. This activates a signal which is sensed by the keyboard encoder. On releasing the key, the plunger comes back to its original position due to a spring action. The connection is also removed. This type of keyboard is bulky, and has wear and tear problem. But it can be serviced easily, when it develops a fault. Figure 11.3(b) shows the block diagram of the keyswitch.

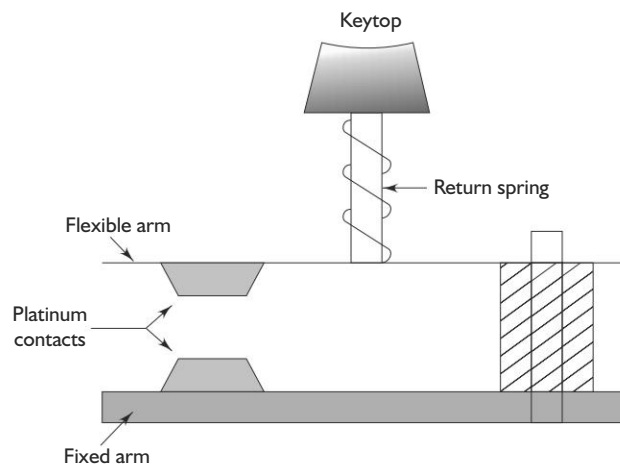


Fig. 11.3(a) Mechanical keyswitch

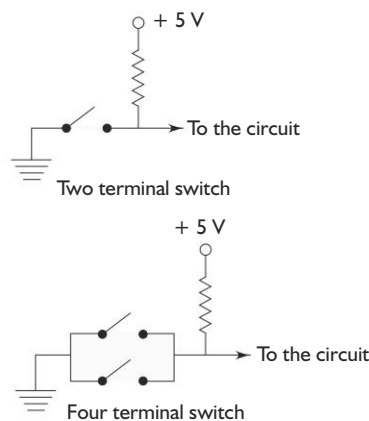


Fig. 11.3(b) Block diagram of a keyswitch

11.2.1.2 Capacitive Keyswitch Keyboard

The capacitance between two conducting plates is affected by key activation. The plates are placed with a small gap filled by a mylar sheet. When a key is pressed, the plunger movement pushes the plates reducing the gap between them. This changes the capacitance between the sheets (plates). The keyboard electronics senses the capacitance change and identifies the key. This type of keyboard is small in size, and more durable. But when it fails, it is not easily serviceable.

11.2.1.3 Membrane Keyswitch Keyboard

In membrane switch keyboard, a number of membrane switches are present below the keys, and there are no springs. On pressing a key, an electric circuit is closed by two metallic contacts. On releasing the key, this circuit is broken. There is less moving parts in the membrane keyboard, which is a silent keyboard.

11.2.2 Keyboard Organization

There are several ways of organizing the keyswitches inside a keyboard. Figure 11.4 shows the simplest method: each key's output is directly connected to the electronics circuits, which results in fast encoding, but requires a lot of hardware circuits. This arrangement is suitable only for small keyboards found in calculators and electronic cash registers. The matrix organization is the popular technique followed in large keyboards used in computers. Figure 11.5 shows a matrix keyboard in which the keys are organised in rows and columns. Each key has a unique set of coordinates: A row number and a column number. When a key is pressed, the keyboard electronics has to determine the pressed key's row number and the column number. This encoding is performed in matrix keyboards by scanning technique. The scanning method uses row as inputs to the matrix and the columns as the outputs from the matrix. It scans the matrix row, by row to determine the key press.

11.2.3 Keyboard Microcomputer

The early PC keyboards use a microcomputer chip, Intel 8048. Some keyboards use a microcontroller such as Intel 8039 or Intel 8031, and external support hardware. In most keyboards, the microcontroller firmware is implemented in the ROM, inside the microcontroller. The functions performed by the keyboard microcontroller are as follows:

1. Scanning the keyswitch matrix
2. Detecting a key press

3. Debouncing
4. Generating a Make scancode
5. Buffering of upto 20 scancodes if the PC is busy, not accepting the scancode.
6. Transmitting the scancode on serial interface
7. Providing bidirectional serial communication for DATA and CLOCK lines
8. Observing handshake protocol for each scancode transfer
9. Performing Power-On Self-Test; the self-test includes checking internal ROM and RAM, and detecting stuck keys
10. Typematic action: repeating the transmission of make scancodes, if a key is pressed down constantly.

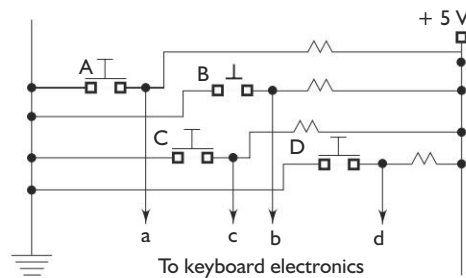


Fig. 11.4 Direct output keyboard

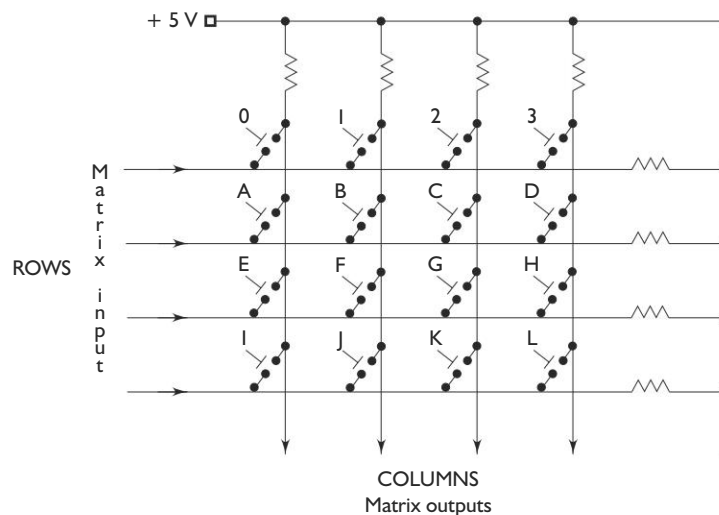


Fig. 11.5 Keyboard matrix



11.3 CRT Display Monitor

The Cathode Ray Tube (CRT) display is a widely used Visual Display Unit (VDU) for the past several years. The CRT display is also called CRT monitor. It is commonly used in all desktop computers, where its huge size and heavy weight is not a problem. In portable computers, Liquid Crystal Displays (LCDs) are used due to its light weight, small size and ruggedness.

Figure 11.6 presents a block diagram of a CRT monitor. The CRT monitor receives video signals from the computer, and displays the video information as dots on the CRT screen. The computer has a CRT controller circuit which works in synchronization with a CRT monitor. The main unit in the CRT monitor is the CRT itself; it is usually called a picture tube. The CRT is an evacuated glass tube with a fluorescent coating on the inner front surface, called screen. An electron gun at one end (neck) emits an electron beam. This beam is directed towards the screen. When the beam strikes the screen, the phosphor coating on the screen produces illumination at the spot where the electron beam strikes. The electron beam is deflected by an electromagnetic deflection in order to produce illumination at various spots on the screen. Horizontal deflection coils deflect the beam in the horizontal direction and the vertical deflection coils deflect the beam in the vertical direction. The illumination caused on the screen exists for a few milliseconds due to the persistence of the phosphor. To create a permanent image on the screen, it is necessary to cause illuminations repeatedly. This is done by scanning the CRT screen with the electron beam. The common method of scanning is called Raster scan. In this method, the electron beam is moved back and forth across the screen. On reaching the extreme right, the beam is brought back to the left. It, then, moves right from the next scan line. On reaching the bottom of the screen, the beam is brought to the top of the screen. To produce an image, the beam is turned on or off. The video information from the computer is used for turning the beam on or off at appropriate places when the beam scans the screen. The HSYNC from the computer provides horizontal synchronization for each scan line, i.e. when the beam starts from the left side of the screen after returning from the right. VSYNC from the computer is used for vertical synchronization for each raster. Colour monitors produce colour images and are more popular than monochrome monitors.

There are two types of images on CRT displays: Alphanumeric displays (or text) and graphics. The alphanumeric display system generally follows the dot matrix scheme for each character generation, as in a dot matrix printer.

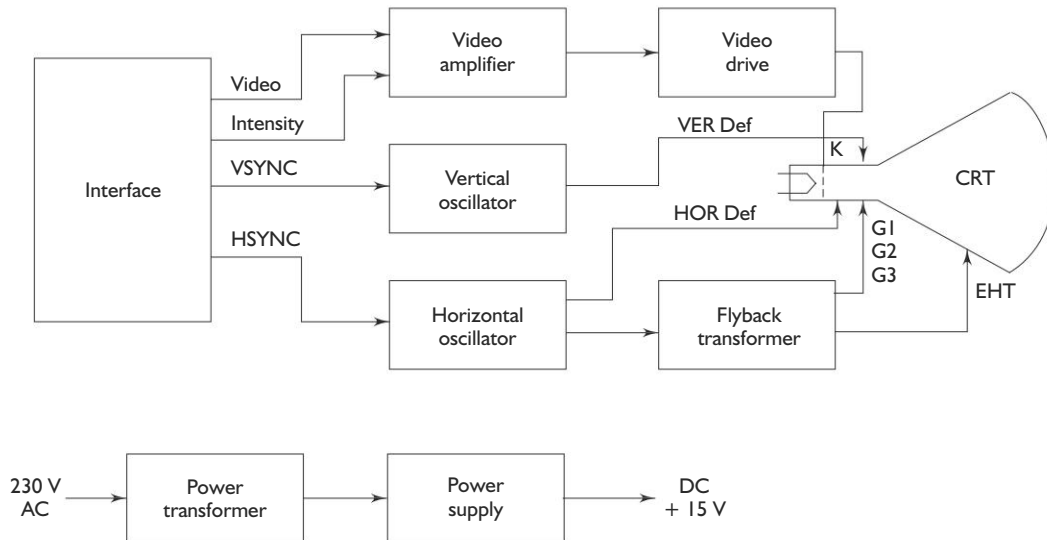


Fig. 11.6 CRT monitor block diagram

There are two methods of interfacing a CRT monitor to a computer:

1. TTL video interface
2. Composite video interface

In the TTL video interface, the computer sends video, HSYNC and VSYNC signals on separate wires, as shown in Fig. 11.7. In the composite video interface, all the three signals, video, HSYNC and VSYNC are mixed, and sent as a composite video waveform on a single wire, as shown in Fig. 11.8. Based on the type of interface incorporated, CRT monitors are classified into two types: TTL monitors and composite video monitors. The modern displays are analog monitors offering excellent resolution, and large number of colour options.

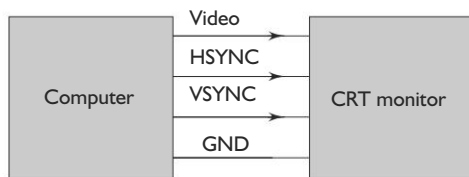


Fig. 11.7 TTL interface

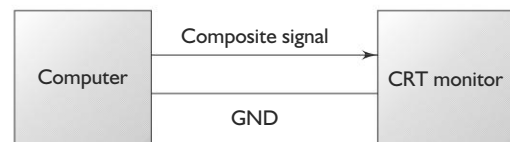
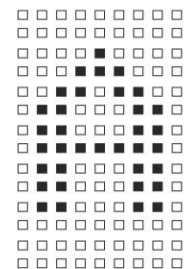


Fig. 11.8 Composite video interface

11.3.1 Character Formation

The CRT Display is used to display numbers, letters and graphics. There are two techniques followed for producing images on the CRT screen: Raster scan and Vector scan. The raster scan technique is universally followed in the CRT displays used in the PCs. In raster scan technique, the horizontal and vertical deflection signals are generated to move the electron beam back and forth across the screen like a raster. The retrace portion of the raster scan pattern is suppressed (blanked); this is achieved by reducing the intensity of the electron beam during retrace. The complete CRT screen can be considered to be made of dots or pixels (picture elements). Any dot can be illuminated by the electron beam, when the beam goes through the dot during the raster scan process, or it can be made invisible by reducing the intensity of the electron beam.

The character formed on the CRT screen is not continuous, but a collection of dots which create the character-like pattern within a matrix of dots. Figure 11.9 shows how the letter is constructed in a 7×9 dot matrix. The character box size is bigger than the actual character matrix size to provide, space between adjacent characters in the same row, space between characters in the adjacent lines, and extra space for lower case alphabets to descend below the normal base line of upper case alphabets. Any alphabet, number or picture to be displayed on the CRT can be constructed by displaying some pixels and blanking out the remaining pixels in the character matrix, according to the pattern to be displayed. The scanning process has to be repeated quickly within the persistence of the phosphor, so as to present a stable display. Typical horizontal sweep or scan rate of the electron beam for CRT monitor is 15.75 kHz and vertical sweep frequency is 60 or 50 Hz. Table 11.1 gives the scan rate for different display adapters.



Character box – 9×14
Character – 7×9

Fig. 11.9 Character generation

11.3.1.1 Colour Monitor

In colour monitors, the screen is coated with a pattern of little circular or rectangular phosphor dots. Three types of phosphor are used on the same screen and these are distributed equally on the screen. The red type phosphor dots glow red when hit by an electron beam, the blue type phosphor dots glow blue, and the green type phosphor dots glow green. The

monitor has three electron beams corresponding to the three basic colors: red, green and blue. Each electron beam illuminates phosphor dot of one color. Any color is obtained by varying the intensities of the red, green and blue electron beams in definite proportions. When all the three beams are of equal intensity, white color is produced.

TABLE 11.1 Display adapters and scan rates

S. No.	Adapter type	Mode	Horizontal scan rate (kHz)	Vertical scan rate (Hz)
1	MDA	—	18.432	50
2	CGA	—	15.750	60
3	EGA	MDA	18.432	50
4	EGA	CGA	15.750	60
5	EGA	EGA	22.000	60
6	PGA	—	30.480	60
7	MCGA	PS/2	31.500	70 or 60
8	VGA	PS/2	31.500	70 or 60
9	SVGA	—	90.000	90

11.3.1.2 CRT Interface

A typical CRT monitor provides three signal lines to a CRT controller: Horizontal Synchronization (HSYNC), Vertical Synchronization (VSYNC) and VIDEO. The horizontal and vertical deflection control signals are generated by the circuits in the monitor electronics and applied to the tube. The horizontal and vertical sweep oscillators of a CRT monitor are free running, and the scanning is usually continuous. The purpose of the synchronisation signals is to shorten or lengthen the existing scan motions to synchronise the display of information via the electron beam. The intensity of the electron beam is modulated by the video input; simple on/off levels produce dots or no dots. Blanking levels are fed to the video input to turn off the electron beam, during the return trip (retrace) of each horizontal scan and vertical frame scan.

11.3.1.3 Composite Video

In some CRT monitors, a single input signal called Composite Video is provided. The composite video signal includes HSYNC, VSYNC and VIDEO. The CRT monitor has the circuitry to separate out the three signals comprising the composite video input. The composite video signal can be sent on a single coaxial cable. This is more convenient and cheaper over long cable runs.

11.3.1.4 Digital vs Analog

Till the VGA was introduced in PS/2, almost all the displays used digital display. A digital signal has two states: ON and OFF. Separate signals are used for colour and intensity. In the analog system, the voltage of the signals controlling the colour beams vary continuously. Only a few lines are needed for the three primary colours. The intensity voltage for each colour can be varied almost indefinitely. In VGA, this gives 256 simultaneous colours out of a maximum of 2,62,144.

11.3.1.5 True Colour

A single bit can make a pixel ON or OFF. To provide Colour shades and depth, several bits are used per pixel. True colour requires displaying a large number of shades in separate distinct colours. To achieve true colour, a display adapter should have large and fast video buffer, a dedicated processor/controller and sophisticated video processing logic.

Depth

Each colour pixel can have a large amount of information about each colour. For instance, a 24-bit system has 8 bits of information about each colour.

11.3.2 Video Bandwidth

The bandwidth of a monitor indicates the range of frequencies that can be handled by the circuits (video amplifier etc.) inside the monitor. It is an important specification provided by the monitor manufacturer. An approximate estimate can be estimated from the resolution (R) and the vertical scan rate (S). Multiply R by S and add some overhead (approximately 0.4 per cent) for retrace. A typical very high resolution monitor provides 200 MHz video bandwidth. A sample calculation is provided below, which can be used for working out the bandwidth requirement of a monitor.

Example 11.1: Bandwidth calculation

Resolution (R): 800×600

Vertical Scan Rate (S): 90 Hz

$R \times S = 43.2 \text{ MHz}$

Overhead = 1.728 MHz

Total = 44.928 MHz or 45 MHz

Variable frequency monitors accept a wide range of frequencies, and can handle more than one type of colour signal encoding. A multi-sync monitor can detect the type of VGA mode being used and adjust itself suitably.



11.4 Printer

The printer is an electromechanical device. It has both electronic circuits and mechanical assemblies. The electronic circuits control the mechanical assemblies. Hence, the electronic circuits in a printer are usually referred to as printer electronics or control electronics. Figure 11.10 presents a simple block diagram of a printer. A computer interface links the printer with the computer. Commands and data from the computer are sent to the printer through this interface. The printer sends its status to the computer through this interface. The printer electronics has necessary circuits to decode the command, generate control signals and activate the print mechanism to print data received from the computer. The mechanical assemblies include print head assembly, print carriage motor, ribbon assembly, paper movement assembly, sensor assemblies etc.

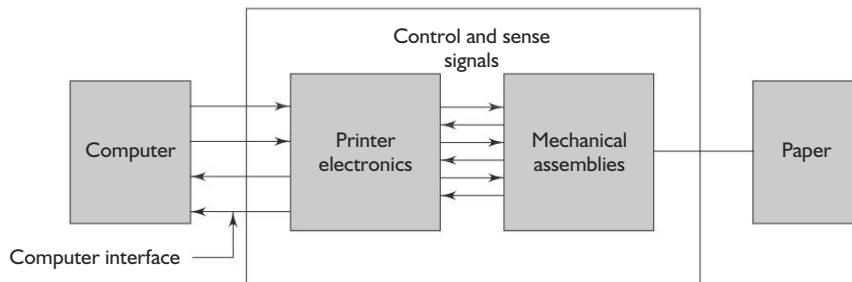


Fig. 11.10 Printer block diagram

11.4.1 Printer Functions

The printer receives data characters from the computer, and prints the characters on the paper. In addition, the printer also receives control characters from the computer. The control characters are not printable characters. They convey some sort of control information to the printer. Some of the control characters widely used are CR (Carriage Return), LF (Line Feed) and FF (Form Feed). CR specifies that the printer head carriage should return to the first print column. Any subsequent data character received will be printed starting from the first column. LF informs the printer to skip one line on the paper. FF instructs the

printer to skip the paper to the beginning of the next page (or form). The printer stationery (paper) is available as continuous sheets folded into pages. Each page is known as a form.

11.4.2 Printer Characteristics

There are a wide variety of printers. They differ in different aspects like performance, price and quality. The main characteristics of the printers are listed below:

1. **SPEED** specified as CPS (Characters Per Second), LPM (Lines Per Minute) or PPM (Pages Per Minute). It indicates how fast a printer works.
2. **QUALITY** specified as DRAFT, NLQ (Near Letter Quality) or LQP (Letter Quality Printer). This implies how good the shape of the printed character is.
3. **CHARACTER SET** indicating the total number of data characters and control characters recognised by the printer.
4. **INTERFACE** specifying whether the printer receives characters from the printer in parallel form (one character at a time) or in serial form (one bit at a time).
5. **BUFFER SIZE** indicating how many data characters can be stacked in the printer buffer memory before printing.
6. **PRINT MECHANISM** specified as impact dot matrix, impact daisy wheel, impact golf ball, electrosensitive dot matrix, thermal dot matrix, band, belt, drum, train, chain, ink jet or laser.
7. **PRINT MODE** specified as serial or parallel.
8. **PRINT SIZE** specified as character size and number of characters per line (number of print columns).
9. **PRINT DIRECTION** specified as unidirectional, reverse, bidirectional logic seeking.

11.4.3 Printer Types

Printers are classified into various types, as shown in Table 11.2. Some types of printers such as drum printer and chain printer are obsolete today, though they were very popular with the older maxicomputers. With current microcomputers, impact dot matrix printers and ink jet printers are the two types widely used. The laser printer is used for high speed applications. The LED printer is similar to laser printer, but offers higher speed and easy maintenance.

Irrespective of the type of printer, the printer has to move the paper into position, print on it, and then move it. The common forms of paper are single sheets, fanfold and roll. The

paper movement is done in one of the many different ways: friction feed, tractor feed, pin feed and sheet feed. The modules inside a printer are:

1. Print-head mechanism
2. Carriage-movement mechanism
3. Paper-feed mechanism
4. Control electronics
5. Interface logic
6. Power supply

TABLE 11.2 Printer types

S. No.	Feature	Classification
1.	Printing Technique	(a) Impact Printer (b) Non-Impact Printer
2.	Printing Sequence	(a) Serial Printer or Character Printer (b) Parallel Printer or Line Printer
3.	Print Quality	(a) Draft Printer (b) Letter Quality Printer (LQP) (c) Near Letter Quality Printer (NLQ)
4.	Print Mechanism	(a) Dot Matrix Printer (b) Daisy Wheel Printer (c) Golf-Ball Printer (d) Drum Printer (e) Band Printer (f) Chain Printer (g) Train Printer (h) Thermal Printer (i) Spark Gap Printer (j) Ink Jet Printer (k) Laser Printer (l) LED Printer
5.	Printer Interface	(a) Parallel Interface Printer (b) Serial Interface Printer
6.	Print Direction	(a) Unidirectional Printer (b) Bidirectional Printer (c) Reverse Printer

11.4.3.1 Impact and Non-Impact Printer

In an impact printer, the character is formed by physical contact (or pressure) of the print head (hammer, pin or font) against an ink ribbon and onto paper. The dot matrix printer, daisy wheel printer, golf-ball printer, drum printer, band printer and chain printer are all impact printers.

In a non-impact printer, there is no physical contact of the head with the paper or ribbon. The laser printer, thermal printer, ink jet printer and electrostatic printer are all non-impact printers.

11.4.3.2 Character Printer and Line Printer

In a character printer, characters are printed one after the other. Only one character can be printed at a time. The character printer is also known as serial printer. The daisy wheel printer and dot matrix printer are examples of character printers.

In a line printer, many characters of a line are printed at a time. To an user, the line printer appears to print a complete line of character in one shot. The drum printer, band printer and chain printer are examples of line printers.

Because multiple print characters are printed simultaneously in a line printer, the speed of a line printer is very high compared to that of a character printer. The speed of a character printer is specified in CPS and that of a line printer is specified in LPM or in PPM.

11.4.3.3 Draft, LQP and NLQ Printers

In a draft quality printer, the print character is formed by closely spaced dots, creating character shape. Though the character is visible, it is not impressive because of the presence (visibility) of dots. The dot matrix printer is an example of a draft printer.

In a LQP, a whole character is formed as a letter (without dots), like a typewriter. It is pleasant to read, and hence, LQP is widely used in office environments. The daisy wheel printer is an example of a letter quality printer.

A NLQ printer also prints characters as patterns of dots. But each character is printed twice. The heads (dots) are offset by a minute distance during the second time. Hence, the character is more impressive.

11.4.3.4 Parallel Interface and Serial Interface

In a parallel interface printer, the printer receives all 8 bits of a character simultaneously from the computer, as shown in Fig. 11.11. Hence, there are 8 data lines (wires) between the computer and printer.

In a serial interface printer, the printer receives the 8 bits of a character one after the other, in a serial fashion, as shown in Fig. 11.12. There is only one data line between the computer and printer. The printer internally assembles one character from the serial bit stream.

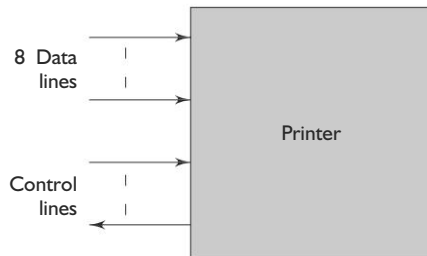


Fig. 11.11 Parallel interface

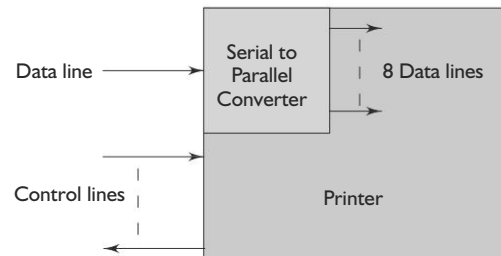


Fig. 11.12 Serial interface

The data transfer rate, i.e. the number of characters transferred per second, between the computer and printer, is more with a parallel interface. Some printers have both parallel and serial interfaces. The user can install/utilize one of the two, as per his choice. The centronics parallel interface is the most popular interface with microcomputers. In this interface, a 36 pin centronics connector is used in the printer. RS232C is the most popular serial interface with microcomputers and minicomputers. Modern printers use USB interface that has been discussed in Chapter 10.

11.4.3.5 Unidirectional and Bidirectional Printers

In an unidirectional printer, the printing is performed only when the print head moves in one direction, i.e., from left to right. No print operation is done when the head returns from right to left. This is similar to a manual typewriter operation.

In a bidirectional printer, printing is performed during both directions of the head movement. If one line is printed when the head moves from left to right, the next line is printed when the print head moves from right to left. The last character in the line is printed first, and then the other characters in reverse direction. The print speed of a bidirectional printer is more than the speed of an unidirectional printer.

11.4.4 Daisy Wheel Printer

In a daisy wheel printer, the print head consists of a circular wheel. There are 96 spokes or character arms on the wheel. Each spoke has a raised character (print block) embossed at

the tip. To print a character, the wheel is rotated until the petal carrying the desired character is in front of the print space on the paper. Then, the solenoid driven hammer strikes the petal, thereby striking the ribbon and paper. After this, the print head is moved to the next column.

The print quality of a daisy-wheel printer is very good. The fonts can be easily interchanged. The speed ranges from 10 to 60 CPS.

11.4.5 Dot Matrix Impact Printer

The dot matrix printer does not print a whole character. Each character is formed by small dots. A matrix, usually a 7 by 5, is followed to create the character pattern of dots. The print head consists of pins arranged in a vertical column. There is a solenoid corresponding to each pin. The character to be printed has dots in certain positions of the matrix. The head moves column by column in the matrix. When the head is in one of the columns of the matrix, all the required dots for that column are formed by striking appropriate pins. Then, the head moves to the next column in the matrix, and the process repeats. When all the columns in the matrix are covered, one character pattern is complete. The characters created in this fashion are of draft quality. By using a greater number of pins (9, 14, 18 or 24) or by printing a line twice with a slight offset for the dots in the second printing, NLQ is achieved with dot matrix printer.

The dot matrix printer can also print graphics. Any figure or pattern can be created by forming dots sequentially, column by column. Without any hardware change, the dot matrix printer can be made to print in any language. Colour printing can also be achieved by a dot matrix printer, using different colour ribbons. The same line is printed in many passes—in different passes, different colour ribbons are selected. The speed of dot matrix printers in the draft mode varies from 100 to 300 CPS. The speed in NLQ mode varies from 25 to 50 CPS. The price of the dot matrix printer is lower than that of a daisy wheel printer.

11.4.6 Golf-Ball Printer

In the golf-ball printer, there is a spherical golf-ball like metallic unit. The different characters are embossed on the golf-ball, as a raised (projecting) type around a sphere. To print a character, the drive mechanism causes appropriate movement of the ball, so that the character to be printed is brought in front of the desired position on the paper. Next, the ball is pushed over the ribbon and the letter (character) is printed on the paper.

The print quality of a golf-ball printer is excellent. The font (character set) can be easily changed by changing the ball. The speed is usually less, around 10 to 15 CPS.

11.4.7 Thermal Printer

Usually, thermal printers use special heat sensitive paper. The print head consists of heating elements. The characters are printed as a matrix of dots. When a particular heating element is switched ON, the corresponding spot on the paper is heated and the spot turns dark. Thus, dots are burnt onto the paper, thereby creating characters or graphics.

In some thermal printers, the paper is a normal one but the ribbon is heat-sensitive. When a spot on the ribbon is heated, a dot of ink is fired on the spot in the paper. A character is formed by a matrix of dots.

The thermal printer is quiet and the print quality is consistent. The running cost is high because of special paper or ribbon.

11.4.8 Laser Printer

Laser printer offers high quality and high-speed printing of images and text data, suitable for desktop publishing. Laser printer technology is based on static electricity similar to photocopier. It uses a xerographic printing process and the image is produced by the scanning of a laser beam across a photoreceptor.

11.4.8.1 Print Image Data

The laser printer prints one whole page at the same instant. Before printing, initially, the image of the page (data) to be printed is created representing every dot on the page, and stored in an internal memory. Hence, the image of an entire page becomes a matrix of small pixels, before print operation starts. To achieve this, the printer electronics has an embedded microprocessor/microcontroller in addition to internal memory. As and when the printer receives the data characters (for a page) from the computer system, the printer electronics generates appropriate pixel patterns and stores them in the internal memory at relevant locations. The printer electronics performs this operation starting from the first character to the last character of a page. Each horizontal strip of dots across the page is known as a raster line or scan line.

Laser printer uses a special command language known as page description language (PDL) which translates the contents of a file into the bitmap images to be printed on the page. There are some popular standards for PDLs: Adobe PostScript (PS), Hewlett-Packard's Printer Control Language (PCL) and Microsoft XML Page Specification (XPS). The source material (data) is encoded in any such PDL. The PDL is a display language designed primarily to generate printed documents. It can be considered both as a display

language and programming language. The printer electronics uses the PDL to generate the bitmap of the page in the memory. Once the entire page has been stored in internal memory, the printer is ready to begin the process of sending the dots to the paper in a continuous stream.

11.4.8.2 Print Mechanism

The basic operation of monochrome (black and white) laser printer is considered here. The print mechanism (Fig.11.13) uses the principle of electrical charge attracting ink. The print mechanism consists of following modules/components:

1. A rotating photosensitive drum
2. A laser light source that is switched ON and OFF by the print electronics, according to the stored pixel matrix
3. A Toner (a dry ink) container
4. A set of rollers for paper feed
5. A set of heated rollers to fuse the toner when the paper passes through them

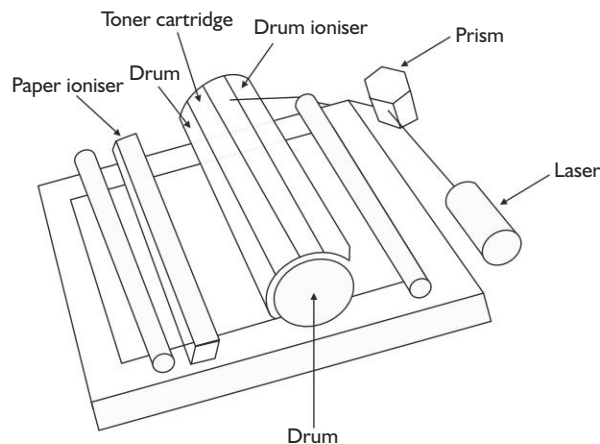


Fig. 11.13 Laser Printer

The electro photographic process in laser printer involves following six steps (Fig.11.14):

1. *Building up an electrical charge on a revolving drum:* A photosensitive surface (photoconductor) is uniformly charged with static electricity by a corona discharge. This step applies a negative charge to the photosensitive drum.
2. *Creating a surface with positive and negative areas:* The charged photoconductor is exposed to an optical image through light to discharge it selectively and forms a latent or invisible image. This step writes the bitmap to the photosensitive drum.

The laser is aimed at a rotating polygonal mirror, which directs the laser beam through a system of lenses and mirrors onto the photoreceptor. This scanning process (Fig. 11.15) is similar to electron beam scanning used in CRT display. The beam sweeps across the photoreceptor at an angle to make the sweep straight across the page; the cylinder continues to rotate during the sweep and hence the angle of sweep should compensate for this movement. The stream of dots turns the laser ON and OFF, to form the dots on the cylinder. The laser beam neutralizes (or reverses) the charge on the white parts of the image, leaving a static electric negative image on the photoreceptor surface.

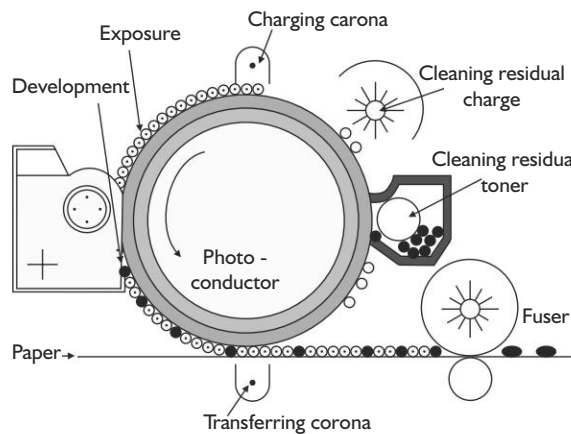


Fig. 11.14 Laser Printer-printing process

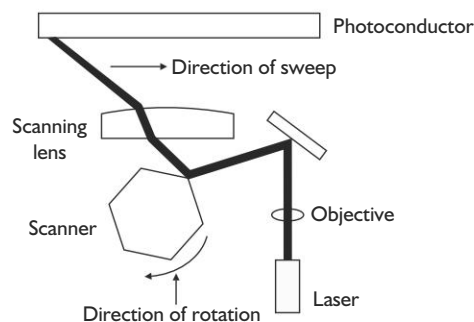


Fig. 11.15 Laser Printer scanning process

3. *Coating with toner:* The surface with the latent image is exposed to toner, a fine powder, which sticks only to the charged areas, thereby making the latent image visible. The charged toner particles are given a negative charge, and are electrostatically attracted to the photoreceptor's latent image, the areas touched by the laser. The negatively charged toner does not touch the drum where the negative charge remains.
4. *Forming the image on the paper:* An electrostatic field transfers the developed image from the photosensitive surface to a sheet of paper. For this, the photoreceptor is pressed or rolled over paper, transferring the image. Higher-end laser printers use a positively charged transfer roller on the back side of the paper to pull the toner from the photoreceptor to the paper.
5. *Fixing the image:* The transferred image is fixed permanently to the paper, by fusing the toner with pressure and heat. For this, the paper is passed through rollers in the fuser assembly where heat (up to 200 Celsius) and pressure bond the plastic powder to the paper.
6. *Cleaning:* The last step is cleaning of excess toner and electrostatic charges from the photoconductor to make it ready for next cycle. When the print is complete, a soft plastic blade removes any excess toner from the photoreceptor and a discharge lamp removes the remaining charge from the photoreceptor.

The six steps of the printing process occur in quick sequence before the drum completes one revolution. The high speed laser printer models can print over 200 monochrome pages per minute (12,000 pages per hour). The fastest colour laser printers can print over 100 pages per minute (6000 pages per hour). Colour lasers need multiple passes in order to mix the different colour toners. A colour printer has three print mechanisms one for each primary colour. Hence a colour printer has three drums over which the paper passes through.

11.4.9 LED Printer

The only difference between the laser printer and the LED printer is the method of exposition or formation of the latent image. There are no moving parts in a LED Printer. The LED printer uses an array of Light Emitting Devices to form the latent image. With this technology, an electroluminescent diode print head polarizes the drum with a very fine light ray, making very small dots. This technology is suited for high resolutions (1,200 or 2,400 dpi).

11.4.10 Inkjet Printer

Inkjet printer is a non-impact (non-contact) printer. Apart from paper, it can print on many types of surfaces. The major advantages of an Ink jet printer are:

1. Fast printing
2. Quiet printing (noise free)
3. Flexible printing (range of print surfaces)
4. Low price

Though the cost of the inkjet printer is cheaper than dot matrix printer, it has the following disadvantages:

1. High consumable cost: Ink cartridge is costly.
2. Not the best fit for occasional (infrequent) usage.

11.4.10.1 Basic Principle

Printing is done by spraying ink (on the paper) as a series of dots, to form an image. An inkjet printer sprays small (50–60 microns in diameter) droplets of ink. The dots are placed, with resolutions of up to 1440×720 dots per inch (dpi). The dots may be of different colours mixed, to get photo-quality images. The print head is a sealed unit. A set of ink nozzles are present as a vertical column (just like the pins in a dot matrix printer). Ink is supplied to each nozzle, from a reservoir. The ‘throw’ of an ink drop from the nozzle, is done whenever a dot has to be formed on the paper. There are two common techniques to eject a drop of ink via the nozzle:

1. By mechanical pressure due to a crystal vibration
2. By the expansion and bursting of an air bubble due to heating of the ink

In many aspects, the operation of the inkjet printer is similar to a dot matrix printer.

11.4.10.2 Inkjet Printers Technologies

There are two main inkjet technologies used to force droplets of ink through a nozzle: Crystal Vibration and Electrical Heating. Three common models are as follows:

1. *Drop-on-Demand Inkjet* (also known as Piezoelectric): A crystal is present at the back of the ink reservoir of each nozzle. The crystal is given an electrical charge due to which it vibrates. Subsequently, when the crystal vibrates inward, it squeezes a dot of ink, out of the nozzle. When it vibrates outward, it pulls-in some ink into the reservoir.

2. *Thermal Inkjet Printer* (also known as Bubble Jet): Tiny heater elements (resistors) are used for every nozzle. The heater, when activated by the control circuit, creates heat which vaporizes ink and hence an air bubble is formed. The air bubble expands in the ink channel. When the bubble bursts (collapses), its force pushes a drop of the ink out of the nozzle. The vacuum created pulls-in more ink from the reservoir. All the nozzles (300 or 600), can fire droplets simultaneously.
3. *Continuous Flow Inkjet Printer* : This is a special type of Inkjet printer used for 'industrial marking' on boxes, bags etc. In this printer, the print head does not move. Instead, the print surface is moved across the print head by conveyor. The ink flies in a straight direction. A source of air pressure and vacuum are required in this method, apart from a big ink reservoir and cleaning filter (for reusing the re-circulated ink). A stream of pressurized ink is broken into droplets. As each drop passes through the head, a varying electrical charge is applied to the drop. The amount of charge applied to the drop depends upon exact position (vertical) required for the drop. (If blank is required, no charge is applied to the corresponding drop.) Then, the drop passes through a set of electrical deflecting plates so that the drop is deflected out of the head, and falls on the print surface. Printing speed is extremely high in the order of 60,000 cycles per second (greater than 1500 cps). A drop which did not receive any charge (blank), goes to a return vent from where it is pushed (by applying vacuum) to a remote ink reservoir for reuse.

11.4.10.3 Inkjet Printer Parts

Print Head—It contains a series of nozzles that spray drops of ink.

Ink Cartridges—These are of following types: (1) separate black and colour cartridges; (2) colour and black in a single cartridge; (3) one cartridge for each ink colour. The cartridges of some inkjet printers include the print head also.

Print head stepper motor—A stepper motor moves the print head assembly (print head and ink cartridges), back and forth, across the paper.

Stepper Motor Belt—A belt is used to attach the print head assembly to the stepper motor.

Stabilizer Bar—The print head assembly uses a stabilizer bar to ensure that movement is precise and controlled.

Stabiliser bar and belt—Paper feed assembly.

Paper tray/feeder—Inkjet printers have either a tray to load the paper or a feeder.

Rollers—A set of rollers pull the paper in, from the tray or feeder, and advance the paper when the print head assembly is ready for another pass. The rollers move the paper through the printer.

Rollers—Paper feed stepper motor—This stepper motor powers the rollers to move the paper in the exact increment needed to ensure that a continuous image is printed.

Power supply

Control circuitry

Interface port(s)—The parallel port is still used, but newer printers use the USB port. Using a serial port or SCSI port is also possible.

11.4.10.4 Operation Sequence

The application program issues the data to the printer driver. The printer driver translates the data into a format that the specific printer can understand. The data is sent by the printer driver to the printer via the I/O interface (parallel, USB, etc.). The printer stores a certain amount of data in its buffer. The buffer size can range from 512 KB to 16 MB depending on the model.

During idle time, the printer performs clean cycle, to clean the print head(s). The printer electronics activates the paper feed stepper motor, which engages the rollers. The rollers feed a sheet of paper from the paper tray/feeder into the printer. A trigger mechanism in the tray/feeder is depressed when there is paper in the tray or feeder. If the trigger is not depressed, the 'Out of Paper' LED glows, and sends an abnormal status to the computer.

The print head stepper motor uses the belt to move the print head assembly across the page. The motor halts for a brief instant, when the print head sprays dots of ink on the page. At the end of each complete pass, the paper feed stepper motor advances the paper a fraction of an inch. Depending on the inkjet model, the print head is reset to the beginning of the page, or reverses direction, and moves back across the page as it prints (according to the print pattern).

Once the printing is complete, the print head is parked. The paper feed stepper motor spins the rollers to finish pushing the completed page into the output tray.



11.5 Special Types of Disk Drives

The Floppy disk drive and Hard disk drive have been popular as standard auxiliary memory (secondary storage) for the past several years, for storing program and data. There are some recently developed new types of disk drives which are widely bought by several users due to some specific advantages. These are: Super floppies (Zip, LS 120, Sony HIFD), and MO-drives.

Super floppies: Unlike rapid achievements in increasing performance in magnetic hard disks, it has been very slow growth of technology in floppy disk ever since the 1.44 MB floppy disk was introduced. Table 11.3 lists the super floppies, all using special 3 1/2 inch media. These are cheap and good for backup. Also, portable PC Card versions of these drives are available.

TABLE 11.3 Different super floppies

Sl. no.	Drive name	Capacity	Remarks
1	Iomega Zip	100 MB	—
2	LS 120	120 MB	Read/Write compatible with 1.44 MB floppy
3	Sony Hi FD	200 MB	Read/Write compatible with 1.44 MB floppy

11.5.1 Zip Drive

The zip drive is a magnetic storage device. Its principle of operation is similar to the floppy disk, but it uses zip cartridges for the media which is twice as thick as a floppy diskette. Its size is 4 inches square and its typical capacities are 100 MB and 250 MB. The zip drive uses variable number of sectors per track in order to have maximum usable disk space. Iomega is the inventor of zip drives. Bernoulli's box is another name for the zip drive. Better reliability than floppy, and better portability than hard disk are the attractions of a zip drive though it is a non-standard device in a standard PC. This provides us with a common standard to move large files and to make back-ups.

The Zip drive can be either mounted as an internal drive in a PC or used as an external drive. There are two types of interfacing: To SCSI and to floppy/parallel port.

11.5.2 MO Drives

In Magnetic Optic drives, the medium is magnetic, but different from a hard disk. To write to it, it is heated to about 300 degrees Celsius. This heating is done with a laser beam. The laser beam can heat a very minute area precisely. Hence, the magnetic head can write in extremely small spots. Thus, writing is done with a laser guided magnet. During reading, the laser beam reads the media. It can detect the polarization of the micro magnets on the media. The MO disks are fast, and extremely stable; almost wear proof. The data life span is 30 years. There are many MO drive variations, but all are very expensive. The popular one uses the MO-technology Sony's recordable MiniDisc.

 11.6 Mouse and Trackball

The mouse and trackball are two identical pointing devices with a single difference in shape.

11.6.1 Mouse

The mouse has become an essential peripheral device for easy operation of a system. The user's interaction with the system is simple and untiring when a mouse is used, compared to keyboard operation. By mere clicks of the mouse buttons, the user does a lot of work. The mouse is supported by almost all the current operating systems.

The mouse is one of the pointing devices which use a combination of hardware and software to control the position of a graphical cursor. The driver software generates the cursor and keeps track of its position. As the mouse is moved, the mouse driver interprets the signals from the mouse and accordingly moves the cursor. Pressing a button on the mouse achieves selection or manipulation of the options given by an application program.

The name mouse has been given due to the similarity of its physical look (shape and tail-like cable) and movement (fast and scurrying) to that of a mouse.

There are three types of mouse:

1. Mechanical mouse
2. Opto-mechanical mouse
3. Optical mouse

In a mechanical mouse, a small, round rubber ball in the bottom touches the mouse pad. When the mouse is moved, the ball rotates. There are two flywheels inside the mouse touching the ball thereby sensing the horizontal and vertical movements. Since the flywheels are connected to a resistive element forming a mesh of fine black cross hairs, as and when the mouse is moved, the ball translates the mouse movements into electrical signals. The mechanical mouse is simple and cheaper, but needs routine maintenance like cleaning. Also the wear and tear makes it dead soon.

In an opto-mechanical mouse, the ball and light sensitive semiconductors are used for generating electrical signals. There is no wire mesh unlike the mechanical mouse. Instead, there are two rollers placed at right angles to each other and touching the ball. At the end of each roller, there is a slotted wheel formed by perforating the edges of the wheels with tiny holes. On one side of the holes, there is an LED and the other side, a photodiode or a phototransistor is present to function as a photosensor. When the mouse is moved, the ball rotates and the rollers rotate the slotted wheels. The light emitted by the LED is alterna-

tively blocked and allowed because of the slots. Accordingly, the photosensor generates a series of pulses. The opto-mechanical mouse has longer life and better reliability. But, it needs periodic cleaning to remove dust and dirt.

In optical mouse, there is no ball or rollers. A special mouse pad with a grid is used. The mouse movement is tracked by LEDs and light-sensitive transistors. Though expensive, an optical mouse is desirable because of its better accuracy and higher reliability. There are no moving parts, and practically no maintenance is needed.

A mouse has two or three buttons. The function of each button is defined by the software. When a button is pressed, the mouse transmits a signal that is interpreted by the mouse driver. The action taken by a system depends on the operating system and application program. There are three gestures associated with mouse: single click, double click and drag.

11.6.2 Trackball

A trackball is a mouse turned upside down. The ball is present on the top, and it is moved with user's fingertips. The trackball is usually bigger than the mouse ball, and hence better resolution is possible in a trackball. The trackball occupies less space, and is very convenient in a car or aeroplane where a notebook PC or laptop usage is very common. Some keyboards have built-in trackballs.



11.7 Modem

The word modem is an acronym for MOdulator Cum DEModulator. A modem is an input/output device used to link a computer to a telephone line so as to communicate with another computer. The telephone system transmits voice and data as analog signals.

When a computer sends data to another computer, the modem takes the digital data from the computer, modulates (transforms) it into analog voltages that can be transmitted over the telephone line. At the receiving end, another modem converts the analog voltage into digital data.

The growth of Internet has made the modem a household device.

11.7.1 Fax-Modem

A fax-modem is a modem with additional capability of sending and receiving facsimiles (faxes). Using a fax-software, a document, stored in a system can be transmitted via the

fax-modem, to another fax-modem or a fax machine. Similarly, receiving a document sent by another user to this system is done by the fax-modem, and the fax software stores it as a file. The file can be printed whenever necessary.



11.8 Scanner

Scanner is a special input device to convert both pictures and text into a stream of data. There are four common types of scanners: Drum scanner, Flatbed scanner, Sheetfed scanner, and Hand-held scanner. The drum scanner is a high-end product in terms of quality and price, whereas the hand-held scanner is on the other extreme.

11.8.1 Drum Scanner

A drum scanner uses a PM (Photo-Multiplier) tube which is a light sensing device. It offers a high sensitivity and good signal-to-noise ratio. The image to be scanned is mounted on a spinning drum. Even shadow information that is not visible to human eye, can be captured by the drum scanner, and transformed to the visible region which improves the image.

11.8.2 Flatbed Scanner

The material to be scanned (drawing or text) is kept to lie on a flat bed of glass in the scanner. A light source and a CCD (Charge Coupled Device) are mounted on a motorised carriage. The CCD enables conversion of light (and shades of light) into electric pulses. The scanning head consisting of the CCD moves across the pictures.

11.8.3 Sheetfed Scanner

In the sheetfed scanner, instead of the head moving over the page, the paper is pulled over the scan head. The small size is its advantage, but improper mechanism can skew the paper being scanned.

11.8.4 Hand-held Scanner

The hand-held scanner is held in hand and moved over the document sliding over it. Low cost and portability are the advantages but poor quality is its drawback.

11.9 Digital Camera

A digital camera captures instant digital images on its internal memory or in internal floppy disk. It is interfaced to a PC and hence, the picture is easily transferred to its hard disk. The operation of the digital camera is similar to that of a conventional camera, in which a lens is used to focus the image to be captured. But, in the digital camera, a CCD is used as a primary sensing element instead of a photographic film. The CCD is an array of tiny phototransistors arranged in a grid. Figure 11.16 shows the block diagram of a digital camera.

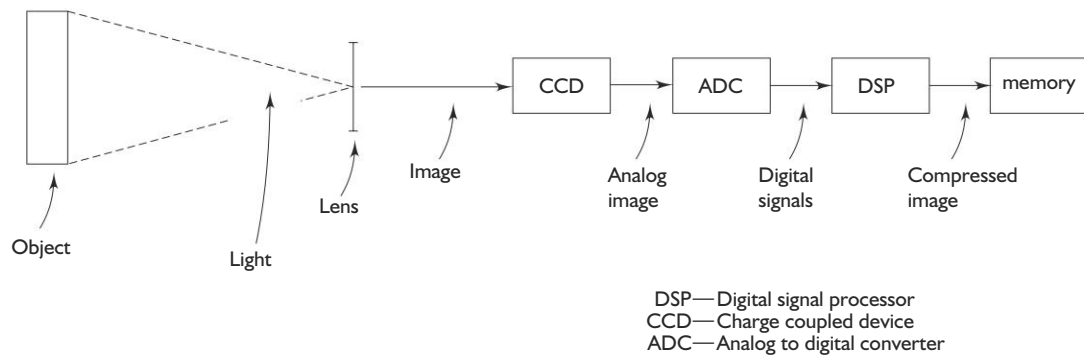


Fig. 11.16 Block diagram of a digital camera

11.10 Special Peripherals

CRT monitor, keyboard, printer, Mouse, Modem, Scanner, CD-ROM, Digital Camera, FDD and HDD are common peripherals used with microcomputers for most of the applications: Data processing, scientific research, word processing etc. For special applications like CAD (Computer Aided Design), CAM (Computer Aided Manufacturing) and DTP (Desk Top Publishing), multi-media, e-learning, video conferencing etc. additional special peripheral devices are used. These peripherals are briefly covered below.

11.10.1 Plotter

The plotter is a graphics output device used to create drawings on paper. There are two types of plotters: pen plotter and photo plotter. The pen plotter is an electromechanical

device. A pen is moved in two dimensions (up/down and left/right) across a paper or film media. Ink pens are normally used in plotters. Colour plotters have multiple pens of different colours. Pen plotters are of many types: drum plotter, microgrip plotter, flatbed plotter etc. In a drum plotter, the drum (a long cylinder) rotates. A pen carriage moves horizontally. In a microgrip plotter, the medium is gripped at the edges and is moved back and forth. In a flatbed plotter, the pen carriage is moved along both the X and the Y axis. To control the motion in short digital steps, stepper motors are used. A photo plotter uses fibreoptics technology to produce the image on dry silver paper.

Plotters are available in different sizes, A4, A3 etc. Though plotters are expensive, they provide excellent quality drawings. Engineering drawings can be prepared quickly and beautifully with plotters.

There are two methods by which a plotter can be interfaced to a microcomputer: via a serial interface, and via a parallel interface.

11.10.2 Light Pen

The light pen is used as an input device in CAD applications. Using the light pen, the user indicates the current active position (of the display) under his consideration, to the computer program. The light pen contains a photo sensor to detect the presence of light. When the tip of the light pen touches a spot on the CRT monitor screen, the light pen is activated. It senses light (when the electron beam scans that spot), and a signal is sent to the microcomputer. The controller which controls the scanning of the electron beam immediately notes down the position of the spot (coordinates) where the light pen is touched. The light pen is connected to the microcomputer through a cable.

11.10.3 Joystick

The joystick is an input device used in simple applications wherein high accuracy is not required. A joystick is used while playing games with computers. The joystick has a lever protruding vertically through the top of the unit. The lever can be tilted at different angles. The joystick is used to control the cursor on the CRT screen which provides a visual indication to the user. The tilt angle of the joystick lever determines the direction of the cursor movement. The rate of speed of cursor movement is proportional to the distance of joystick lever from the vertical position.

11.10.4 Digitiser or Graphic Tablet

The digitiser or graphics tablet consists of a flat surface like a drawing pad. A pointing device is used along with this. A rough sketch can be converted to a fine drawing by the transferring of point and line locations to the computer. This is done on an X–Y line coordinate system. The pointing devices used with the digitiser are many: a stylus or pen, a push-button cursor or puck, a power module or a console and a menu.

A grid pattern of horizontal and vertical lines is present below the flat surface of the digitiser. These lines detect electrical pulses at X–Y coordinate locations where a stylus or puck points. The digitiser is a popular input device in CAD applications. It is available in many sizes.

11.10.5 Touch Pad

The touch pad is a substitute for the mouse as a pointing device. A touch pad works by sensing the operator's finger movement and downward pressure. It has become the popular cursor controlling device in laptops, PDAs, point-of-sale terminals, and information kiosks and other touch screens that require a thin and easy-to-use pointing device. Touchpad can be used to provide screen navigation, cursor movement, application program control, and a medium for interactive input.

The touch pad consists of multiple layers. The top layer is the pad that is touched by fingers. Below the pad, there are layers containing horizontal and vertical rows of electrodes that form a grid. Below these layers, there is an electronics circuitry to which the electrode layers are connected. The layers with electrodes form an ac circuit. As the finger approaches the electrode grid, the circuit path is interrupted and the change in current is sensed by the circuitry. The initial location where the finger touches the pad is stored, and subsequent finger movement is related to that initial point. Some touch pads sense single or double taps of the finger at any point on the touch pad. Other touch pads have two special places where applied pressure corresponds to clicking a left or right mouse button.

Another type of touch pad uses capacitive technique to sense the presence and absolute position of a finger on the pad's surface. It measures capacitance changes, due to a finger's proximity, among electrodes in a grid present below the insulating surface of the pad. From these measurements, it can find out if a finger is touching the pad, and determine the finger's absolute horizontal and vertical position on the surface.

11.10.6 Flat Panel Displays

Flat panel displays are lighter and thinner than CRT displays. The CRT's drawbacks are High power consumption, Low resolution, Harmful electromagnetic radiation and Heaviness. The merits of flat panel monitor are its clear digital picture, small footprint, and light weight. Flat panel displays are based on several technologies. They are divided into two types: Volatile or Static.

11.10.6.1 Volatile Displays

Volatile displays need periodic refreshing of pixels to retain the image. Otherwise, the pixels will gradually lose their coherent state, and the image will “fade” from the screen. The following are some examples of volatile flat panel displays: Plasma displays, Liquid crystal displays (LCDs), Light-emitting diode displays (LED), electroluminescent displays (ELDs), Surface-conduction electron-emitter displays (SEDs) and Field emission displays (FEDs) or Nano-emissive displays (NEDs).

11.10.6.2 Static Displays

Static flat panel displays use materials whose colour states are bistable. Hence, the image does not require any energy to maintain, but requires energy to change the image. This results in a much more energy-efficient display. Bistable nematic liquid crystal display is one such type. Bistable flat panel displays are increasingly used in outdoor advertising and electrophoretic displays in e-book products.

11.10.7 Graphics Accelerator

The wide spread use of graphical applications, especially multimedia applications, has demanded development of graphics accelerators. A basic video controller (Fig.11.17) is based on a LSI chip such as 6845 that is a programmeable CRT controller chip. The video controller has video buffer memory with dual port access. The message to be displayed is stored in the video buffer by the CPU (program). The CRT controller reads the data from the video buffer, and generates corresponding display dot patterns. The dot patterns are serialized as dot signal and mixed with relevant timing signals to generate the video signal for the CRT. Advanced display controllers such as VGA and SVGA offered better resolution, but still not fast enough to meet the speed requirements of modern graphics applications due to following bottlenecks: graphics memory bandwidth, communication between host computer and display controller and monitor refresh.

A graphics-accelerator is an advanced display controller whose basic function is to generate and supply images to a display device. In addition, it accelerates rendering of 3D scenes and 2D graphics, video capture, and more graphically demanding applications such as PC games. A graphics processing unit (GPU) offloads 3D graphics rendering from the CPU. It is used in embedded systems, mobile phones, personal computers, workstations, and game consoles.

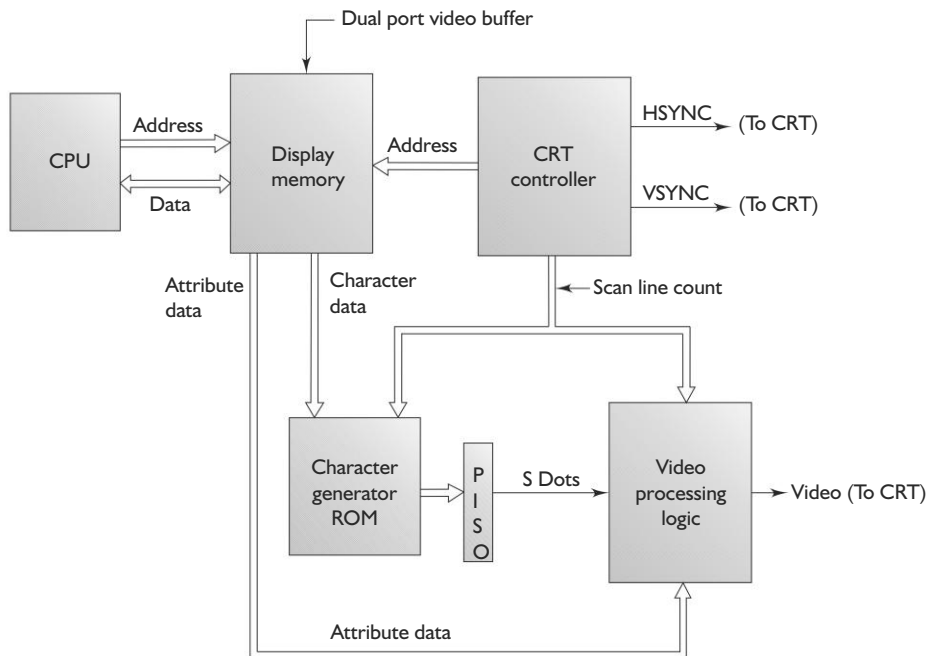


Fig. 11.17 Principle of alphanumeric display

Graphics accelerator is a coprocessor that assists in drawing graphics. Graphics Accelerator contains its own dedicated processor optimized for accelerating graphics to achieve high performance levels. It is a special purpose processor for computing graphical transformations. It also relieves the computer's CPU, so that it can spend its time to perform other non-display actions, while the graphics accelerator is handling graphics computations. The processor is designed specifically to perform floating-point calculations, which are fundamental to 3D graphics rendering. Fig. 11.18 gives the organization of a video accelerator circuit board.

VRAM: Graphics accelerator has internal memory, a special type of video RAM (VRAM), for storing graphical representations. The capacity of memory determines the

resolution and number of colours that can be displayed. The VRAM is a dual port memory permitting both the video circuitry and the processor to simultaneously access the memory through different ports.

The RAMDAC (Random Access Memory Video Digital-to-Analog Converter) converts the digital images (video data) into analog signals (Red, Green and Blue). It also provides horizontal and vertical synchronization signals for use by CRT displays that use analog inputs. Current LCDs, plasma displays and TVs are mostly digital, and hence do not require a RAMDAC. Certain legacy LCD and plasma displays feature analog inputs only. These require a RAMDAC, but they reconvert the analog signal back to digital before displaying it.

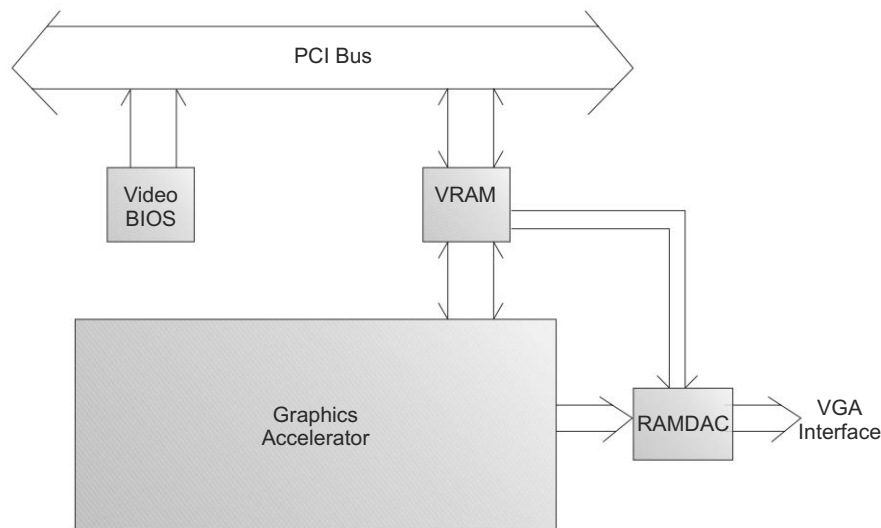


Fig. 11.18 Block diagram-video accelerator

11.10.8 Cable Modem

Cable modem is a modem used for Internet access through cable TV lines. The coaxial cable used for cable TV has a bandwidth of 750 MHz or more (much higher than telephone lines). The total bandwidth is generally divided into 6 MHz bands using frequency division multiplexing. Each cable TV channel needs only 6 MHz bandwidth, and hence, there is excess bandwidth available. Two bands can be used for user's Internet access: uploading and downloading. Thus, a cable modem gives extremely fast access to the Internet and is mainly used for broadband Internet access. The cable modem bridges Ethernet frames

between a customer LAN and the coaxial cable network. A splitter (Fig.11.19a) inside the cable modem directs the TV bands to the TV set and the Internet access bands to the computer's network adapter.

A cable modem (Fig.11.19b) contains following modules/components: tuner, demodulator, modulator, media access control (MAC) device and processor.

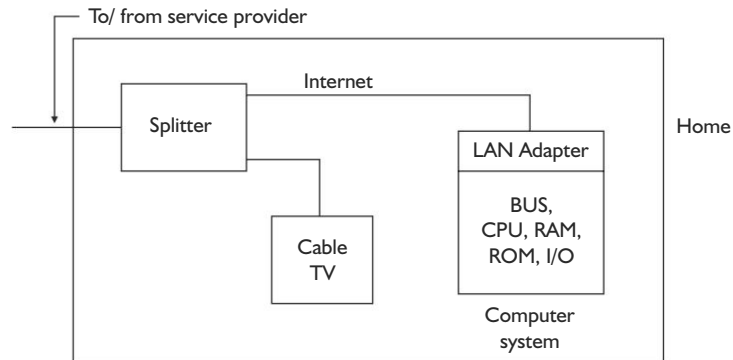


Fig. 11.19(a) Concept of cable modem

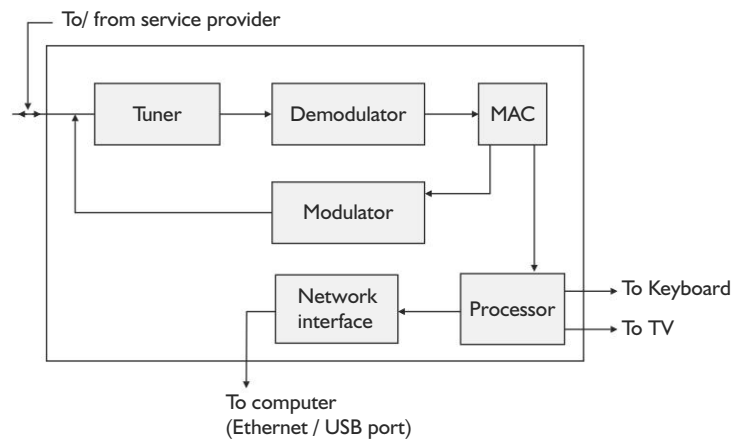


Fig. 11.19(b) Block diagram of cable modem

SUMMARY

Peripheral devices enable interaction between the user and computer. The keyboard is an input device. When a key is pressed, the keyboard electronics transmits a code to the CPU. The keyboard electronics also handles key roll-over problem and key switch bouncing. Mouse and trackball are pointing devices that support graphical user interface. The CRT monitor is an output device that displays the video information. An electron beam strikes the CRT screen causing the illumination. The scanning process involves repeated travel of the beam across the screen creating permanent image.

The printer is a popular output device that provides a hard copy output visible to the eyes. It receives characters from the computer and prints them on the paper. Dot matrix printer, laser printer and inkjet printer are three popular types. The laser printer and inkjet printer are non-impact printers, whereas the dot matrix printer is an impact printer. In an inkjet printer, printing is carried out by spraying ink as a series of dots to form an image.

A modem supports remote communication. Scanner, Digital camera, Plotter, Light pen, Joy stick, Digitizer and Touch pad are some special peripherals used in different applications.

REVIEW QUESTIONS

1. A computer system does not respond to a key press. The service engineer replace the keyboard, but the problem remains unsolved. He concludes that the software (OS/driver) is faulty, which is not true. Detect the problem.
2. Some users are not comfortable with typing on a keyboard. Suggest an alternate solution that does not require the keyboard.

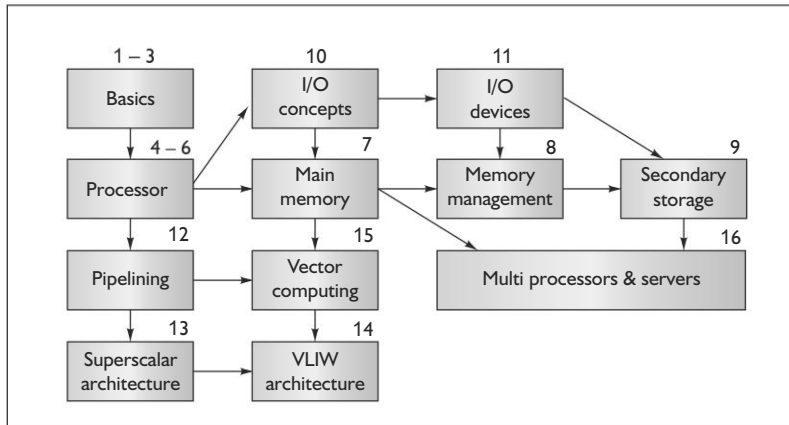
EXERCISES

1. Calculate the bandwidth of a monitor with a resolution of 720×350 and vertical scan rate of 50 Hz.

CHAPTER 12



RISC, PARALLELISM AND PIPELINING



Chapter Flow Diagram

OBJECTIVES

- « RISC Architecture
- « Overlap and parallelism
- « Multiple functional units
- « Pipelining
- « Hazards

KEYWORDS

Overlap	Pipelining
Parallel Processing	Pipeline Hazards
Instruction Level Parallelism	Structural Hazard
Processor Level Parallelism	Data Hazard
Flynn's Classification	Operand Forwarding
SISD (Single Instruction Stream, Single Data Stream)	Delayed Branching
SIMD (Single Instruction Stream, Multiple Data Stream)	Superscalar Architecture
MISD (Multiple Instruction Stream, Single Data Stream)	Super pipelining
MIMD (Multiple Instruction Stream, Multiple Data Stream)	Dynamic scheduling
Vector Processing	Global Resources
Array Processing	Local Resources
	Control Hazard
	Bubble
	Register windows

12.1 Introduction

Achieving high performance from a computer is one of the main goals of a computer architect. Implementing concurrency in the organization of a computer enhances its performance, as multiple operations are simultaneously performed. As discussed in Chapter 1, several techniques of concurrency are available. Providing an overlap or parallelism within a computer is a standard technique, that enhances the performance of the computer. Another technique, that is followed is multiple processors within a computer system. This chapter discusses concurrency in uniprocessor, excluding superscalar architecture which is dealt with in Chapter 13. This chapter provides a detailed study of the pipelining concept and the associated design issues.

Pipelining increases CPU performance, by overlapping execution of multiple instructions. In addition to pipelining, the concept of RISC and the techniques used in RISC processors are discussed in this chapter. The pipelining concepts discussed in this chapter form the foundation for the superscalar architecture covered in chapter 13.

12.2 RISC Systems and CISC Drawbacks

RISC is an acronym for Reduced Instruction Set Computing. RISC is a processor design philosophy that has removed many CISC features, which were occasionally used by the programs. The basic philosophy of the RISC architecture is based on the 'careful allotment' of the processor resources to the frequently used operations. Chapter 3 briefly reviewed the CISC and RISC concepts. A CISC system supports a wide range of instructions in order to provide powerful object codes. This strength of the CISC, is also its weakness. With advancements in the memory technology, the performance of the CISC systems was bogged down by the presence of too many powerful but occasionally used features. Table 12.1 summarizes the criticism made by the RISC advocates against the CISC architecture and the remedies offered by the RISC architecture.

TABLE 12.1 CISC Issues and RISC Solutions

Sl. no.	CISC feature	Motivation/merit	Drawback/criticism	Solution in RISC
1	Uses some complex addressing modes	Short object code is generated by compiler; saves memory space; less number of instruction fetch	Needs heavy hardware circuitry in CPU; in practice, complex addressing modes are used rarely.	No complex addressing mode; manage with simple addressing modes

(Contd.)

S. no.	CISC feature	Motivation/merit	Drawback/criticism	Solution in RISC
2	Has limited number of registers	Reduces CPU hardware	Instruction execution time is more since most operands have to be fetched from main memory	Provides large number of registers; VLSI technology permits this
3	Huge number of instructions in the instruction set	Flexibility to the compiler	Instruction decoding is complex; more circuitry	Supports limited number of instructions
4	Non-uniform instruction length	Flexibility to compiler/assembly programmer	Instruction decoding is complex; more circuitry	Uniform length for all instructions
5	Inclusion of powerful instructions (opcodes)	Flexibility to compiler/assembly programmer; short object code, less memory space, less instruction fetches	Instruction decoding is complex; more circuitry	All are simple instructions
6	Mostly micro-programmed control unit	Easy control unit design	Slow instruction execution due to fetching of microinstructions from control memory (ROM)	Uses hardwired control unit

Compiler developers usually put more effort on optimizing the frequently occurring items, and neglect the rare ones. The absence of the complex addressing modes in the architecture causes the reduction of hardware in RISC. These are substituted by a sequence of simple instructions. Though this appears to reduce performance, the saved resources can speed-up the frequently occurring cases, resulting in an overall enhanced performance. For finding out ‘frequently’ used features, the designers analyzed a large number of source programs and their object codes. This research pin-pointed the frequently occurring types. Table 12.2 summarizes the important observations and conclusions.

TABLE 12.2 CISC program behavior and RISC goals

Sl. no.	Item studied	Characteristics/ behavior observed	Effect	Possible approach in RISC/goal
1	High level language statements— procedure calls	Rarely used	Long object code; large number of memory accesses during execution	Need some technique to reduce execution time for procedure calls
2	High level language statements— loops and conditionals.	Frequently used	–	Time taken for branching should be minimized
3	Operands	Majority of the operands are scalars, either parameters of procedures, or local variables	Stored in main memory; increases memory accesses	A strategy that can provide fast access to scalar variables is needed
4	Addressing modes	memory direct addressing is rare	Needs significant hardware logic	Replace by indexed addressing
5	Parameters and local variables	Most procedures have very few arguments and local variables; depth of procedure calls: negligible percentage of programs nested deeper than 8	–	The processor should make procedure calls with few local variables and parameters as fast as possible

12.3 Techniques of RISC Systems

The following are standard features of most of the RISC processors:

1. All instructions perform simple operations.
2. All instructions are of uniform length which simplify instruction decoding.
3. Instruction format is just one type; or maximum of two/three types
4. The instruction set is orthogonal—there are no restrictions about which operations are permitted with different addressing modes.
5. Just one or two, addressing modes; complex modes are replaced by a sequence of simple arithmetic instructions.

6. The architecture is load-and-store; only load and store instructions can access memory. All other instructions operate on registers. A dedicated unit handles the load and store operation, relieving the integer ALU.
7. Mostly integer data types are supported (though floating point data are also included to speed-up the multi-media applications).
8. A homogenous register set allows any register to be used in any context, simplifying the compiler design.

Due to reduced circuitry, a RISC processor needs few transistors for the core logic. This allows the designers to include several additional features on-chip:

1. Increasing the internal parallelism with pipelining and multiple functional units
2. Internal cache memory
3. Adding internal I/O ports and/or RAM, timers etc. that are needed in microcontrollers
4. Adding internal vector (SIMD) processor

The three new computer projects which established the foundation for the RISC architecture are:

1. RISC I and RISC II at University of California, Berkeley
2. MIPS at Stanford University (1981)
3. IBM 801 at IBM (1975)

Table 12.3 compares RISC I and RISC II systems.

TABLE 12.3 Summary of early RISC systems

Sl. no.	System name	Year	Major features	Special remarks
1.	RISC-I	1982	44,420 transistors, 32 instructions	Three basic addressing modes; 138 registers, no floating-point instructions
2.	RISC-II	1983	40,760 transistors, 39 instructions	Similar to RISC-I; 2 stage pipelined CPU; arithmetic operations are add, subtract, AND, OR, XOR, and shifting.

12.3.1 Branch Delay Slot

A *branch delay slot* (discussed in later sections) is an instruction space immediately following a jump or branch instruction. The instruction in this space is executed whether or not the

branch is taken; the aim is to delay the effect of the branch. This instruction keeps ALU busy for some additional time, which is needed to perform a branch. However, though present in the early RISC processors, most of the modern RISC processors do not support this feature.

12.3.2 Register File and Overlapping Window

A CISC processor has a limited number of registers and the program can use all of them. A good amount of the system's time is wasted in saving and restoring the register contents,

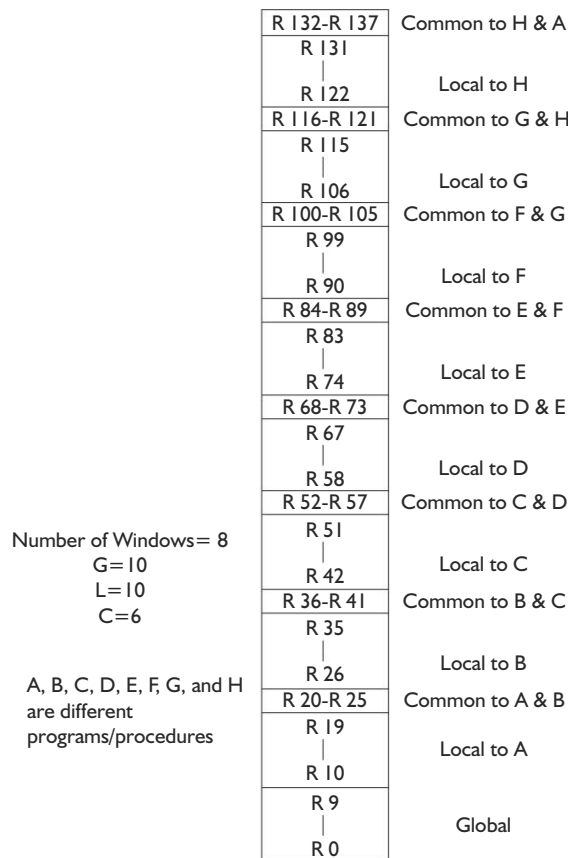


Fig. 12.1 Register windows in RISC II

during the program calls and returns. The technique followed in RISC systems is known as the 'Register Windows'. It eliminates the need to save/restore the registers when calling/returning from the programs. This speeds-up the procedure calls; a call can be handled by moving the pointer, to the currently used set of registers.

For example, the RISC II has 138 registers. But, a given program cannot access all of them, at the same time. There are three types of registers: global, local and common. The first 10 registers are global i.e. they are accessible to all the procedures. The remaining registers form a stack. When a program calls a new procedure, a window of 22 registers from the top of the stack, is allotted to the called procedure. This allows fast passage of the parameters i.e. the calling program simply puts the parameters and return value, in the appropriate registers. The called program simply reads from the common registers. There is no need to save/restore the contents of registers when calling/returning from procedures/functions. These register windows overlap as follows:

- Bottom 6 registers are common with calling procedure.
- Middle 10 registers are local to the called procedure.
- Top 6 registers are common with any new procedure that may be called.

This feature is shown in Fig 12.1. The capacity (R) of the register file is related to the number of windows (W) as follows:

$$R = G + (L + C)W$$

where L is the number of local registers in each window, and C is the number of registers common to two windows, and G is the number of global registers.

A Common Window Pointer (CWP) is used to keep track of the windows. A program 'call' activates a new register window by incrementing the CWP and a 'return' decrements the CWP. If the program has a large number of arguments/local scalar variables, then they are stored in the main memory. Only when a program nesting is deeper than 8, it is required to save the register file in the memory.



12.4 RISC Architecture in Modern Processors

RISC processors are used in servers, high-performance desktop systems, multi-media systems, embedded systems etc. Due to the small size, the RISC processor is readily suited as a low-power "embedded" CPU used in various devices. Similarly, RISC processor is the most preferred choice for high performance workstations. Sun SPARC station is a highly successful RISC workstation.

12.4.1 Modern RISC Processors

Some popular RISC processors used in desktop systems and server system are as follows:

1. Digital Equipment Corp.'s 'Alpha'
2. 'SPARC' from Sun Microsystems
3. 'PowerPC' jointly developed by IBM and Motorola Inc
4. 'MIPS Rxxxx series' from MIPS Technologies Inc
5. Hewlett-Packard Co.'s 'PA-RISC'

Presently, MIPS does not manufacture the stand-alone microprocessors. They supply *core designs or embedded cores* (microprocessor designs) to other developers, to use in special chips (e.g. set-top boxes). Such processor designs are called as the *IP cores* (intellectual property). This reduces the design time for other companies.

SPARC is originally a Sun's architecture, presently maintained by the SPARC international. SPARC is based on the Berkeley RISC project and it uses similar register stacks. Alpha is a 64-bit architecture. The PowerPC and SPARC initially supported a 32-bit architectures which was later extended to 64 bits.

14.4.1.1 Addressing Modes

The addressing modes of these processors differ as regard to the indexed addressing. The Alpha has only one: base register + offset. SPARC has two modes: base register + offset; and base register + index register. PowerPC supports the same modes as the SPARC along with modes which can simultaneously update the index register (i.e. increment/decrement).

12.4.1.2 Registers

All the three processors support 32 general purpose registers. They have 32 floating-point registers, and support IEEE standard floating point. They support four different instruction formats:

- register-register (i.e. arithmetic/logical);
- register-immediate (i.e. data transfer – loads and stores);
- conditional branch;
- unconditional branch and call.

SPARC supports the delayed and annulling branches, with one branch delay slot. Alpha and PowerPC do not support these.

12.4.2 CISC and RISC Convergence

The differences between the CISC and RISC architectures have gradually vanished. Both architectures have adopted each others strategies. The CISC processor designers adapt the original RISC ideas such as pipelining and cache memory. Similarly, certain 'non-RISC' concepts are used in the RISC processors to meet the present day requirements. For example, the early RISC processors did not support floating-point data. But the modern processors support the floating-point data types. This is due to the extensive usage of the graphics-based applications which demand high performance.

12.5 Performance Enhancement Strategies

A computer's performance is measured by the time taken for executing a program. The program execution involves performing instruction cycles, which includes two types of activities:

1. *Internal microoperations* performed inside the hardware functional units.
2. *Transfer of information* between different hardware functional units.

The designer can follow two different approaches for enhancing the performance of a microprocessor:

1. Increasing the clock frequency.
2. Increasing the number of operations performed during each clock cycle.

Increasing the clock frequency depends upon the IC technology.

For performing more operations per clock cycle, two strategies are followed:

1. Providing multiple functional units in single processor chip.
2. Executing multiple instructions concurrently.

As discussed in Section 1.12 (Chapter 1), the performance of a computer (i.e. the time taken to execute a program) depends on three factors:

1. Number of instructions executed
2. Average number of clock cycles required per instruction
3. Clock cycle time

Their relationship is:

$$T = \frac{N(\text{ie}) \times \text{CPI}}{F}$$

As mentioned earlier, reducing the clock cycle time is not an architectural technique, but depends on the IC technology. Research and newer inventions in this area continue.

Various performance improvement strategies are used. The objective is to increase the number of operations executed in a given time. Two basic strategies are followed:

1. *Overlap*: Splitting a task into multiple subtasks that can be performed in an overlapped manner.

2. *Parallelism*: Executing more than one task in parallel.

The *Overlap* strategy uses concurrent operations by heterogeneous functional units whereas the *Parallelism* uses concurrent operations by homogeneous functional units. Figure 12.2 depicts the commonly used overlap techniques. Figure 12.3 illustrates various techniques of parallelism. Broadly speaking, there are two levels of parallelism:

1. Instruction level parallelism (ILP)
2. Processor level parallelism

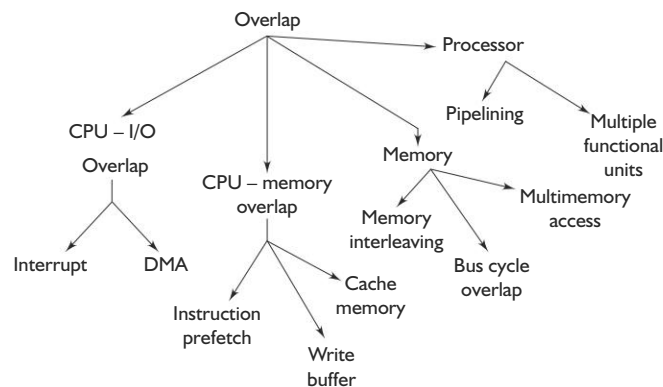


Fig. 12.2 Overlap techniques

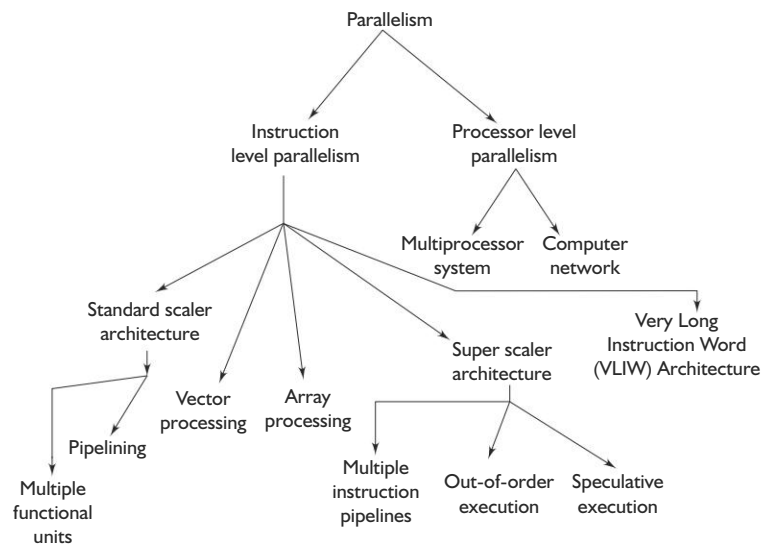


Fig. 12.3 Techniques of parallelism

In the *Instruction level parallelism*, parallelism is applied inside a single processor. It is invisible to programmers. Either the system software (compiler), or the processor hardware can detect the presence of parallelism and exploit it by appropriate techniques. Traditional architectures use various techniques to extract instruction-level parallelism from the instruction stream of an application program, and enhance the performance. Some compilers can restructure the object code, so that the processor hardware encounters ILP directly in the object code. The vector and array architectures take a different approach and exploit the advantage of data parallelism. In *Processor level parallelism*, there are multiple processors that share the workload (tasks). This can be accomplished either by using more than one processor inside a single computer system (multiprocessor system), or distributing the tasks amongst multiple computer systems linked either as a cluster or a network.

12.5.1 Levels of Parallelism

Several levels of parallelism can be achieved in a computer. There are five levels of widely-practised parallelism in a computer. These are:

1. Job level parallelism
2. Program level parallelism
3. Instruction level parallelism
4. Arithmetic and bit-level parallelism
5. Processor level parallelism

Job level parallelism can be achieved by structuring a job (or several jobs) as multiple independent tasks in a multi-programming system. As discussed in Chapter 1, the CPU and the I/O sub-systems in a multi-programming system function in parallel, facilitating the concurrent execution of the multiple tasks by a single CPU. Two different goals can be accomplished:

- (a) CPU's idle time during I/O operation for a task is eliminated to the extent possible by switching the CPU to some other task.
- (b) Multiple tasks progress concurrently.

Program level parallelism can be achieved when a program is divided into multiple parts that are executed by the multiple processors or multiple functional units. Parallelism at the program level is practised in two forms:

- (a) Independent sections of a program
- (b) Individual iterations of a loop

Instruction level parallelism is invisible to programmers. But, the processor can detect it and take appropriate steps. Some special compilers take necessary preparatory steps to restructure the program, so that the processor can encounter instruction level parallelism in the object code. In some cases, specific modes of programming are followed by the user to simplify the compiler's task.

At the instruction level, there are two different options:

- (a) Overlapping execution of individual instructions.
- (b) Overlapping execution of micro-operations of every instruction.

Arithmetic and bit level parallelism is implemented by the designers in the ALU designs. For example, a 16-bit addition can be carried out in the following ways:

1. Adding the 16 bits of the operands using a 16-bit parallel adder; this takes a single clock cycle.
2. Cascade four 4-bit parallel adders of ripple carry adder type (Fig. 4.33); this takes four clock cycles.
3. Cascade four 4-bit parallel adders of carry look ahead type (Fig. 4.33, 4.36 or 4.37); this also takes four clock cycles.
4. Use a serial adder (Fig. 4.30); this takes 16 clock cycles.

The *Processor level parallelism* is achieved by using multiple processors in a computer system. The processors work in parallel. While designing a computer system with multiple processors, the computer architect can follow two different approaches:

1. Using a small number of powerful processors.
2. Using a large number of simple processors.

The second approach is known as *Massively parallel* and it reduces the burden of program development on the multiprocessors.



12.6 Classification of Parallelism

Parallelism is classified by using Flynn's classification of computer organization into four different types. Figure 12.4 illustrates the four types which are as follows:

- Single Instruction stream, Single Data stream (SISD)
- Single Instruction stream, Multiple Data stream (SIMD)
- Multiple Instruction stream, Single Data stream (MISD)
- Multiple Instruction stream, Multiple Data stream (MIMD)

The capability of the processor varies in these different types. The execution unit is indicated as a processor (PR) since it is more complex than a traditional execution unit.

The SISD is a traditional uniprocessor. There is a *single control unit* and a *single execution unit*. Hence, it has one instruction and one data stream.

The SIMD has *one control unit* that handles *multiple execution units*. Each execution unit has a separate data stream. The array processor and Multi-Media Extension (MMX) in Pentium are examples of this type. Basically, while executing a single instruction, simultaneously multiple datapaths operate on multiple data elements.

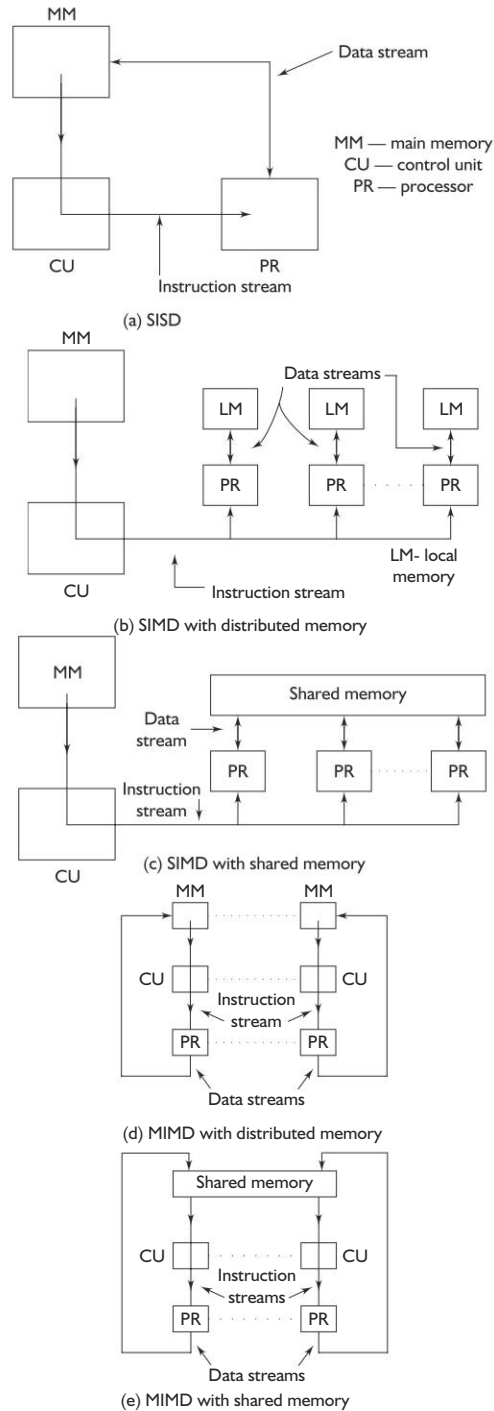


Fig. 12.4 Flynn's classification of computers

The MISD involves *multiple control units* but *a single execution unit*. Such a system is not practically feasible. The MIMD refers to *multiple control units* and *multiple execution units*. Multiprocessors and parallel processors are examples of this type.

12.6.1 Scalar, Vector and Superscalar Processors

A simple *Scalar processor* performs only one arithmetic operation at a time—it executes one instruction at a given time and each instruction has only one set of operands. In a *Vector processor*, each instruction has multiple sets of operands such as elements of two arrays, and the same operation is carried out simultaneously on different sets of operands. Hence, a *vector processor* performs multiple arithmetic operations simultaneously, on different operands. In a *Non-pipelined (sequential) scalar processor*, there is no overlap of successive instruction cycles. In a *Pipelined processor*, at any given point of time multiple instructions are at various stages of instruction cycle simultaneously in different sections of the processor. Hence, the processing of multiple instructions are overlapped in a pipelined scalar processor. An *Array processor* has multiple execution units that operate simultaneously on multiple elements of a vector.

A *Superscalar processor* is a scalar processor that performs multiple instruction cycles simultaneously for successive instructions. Two or more instructions are initiated simultaneously, and in a single clock cycle, same operation (fetch, decode, execute etc.) is executed on two or more consecutive instructions in parallel. To achieve this, multiple functional units are present in the processor.

Though the term ‘superscalar’ appears recently only, there have been some early superscalar processors too, such as CDC 6600 and IBM System/360 Model 91. Usually, superscalar architecture is implemented by multiple pipelines operating in parallel, on consecutive instructions in a single program, whereas a multiprocessor system operates on multiple programs simultaneously. Table 12.4 identifies some well-known processors and their classifications.

Another new classification of processor architecture is Very Long Instruction Word (VLIW) that is a refinement over standard superscalar architecture. The VLIW processor is discussed in Chapter 14.

TABLE 12.4 A sample of processors

Sl. no.	Processor/system	Types	Additional remarks	Flynn's type
1	IBM 1401	Scalar, sequential	Mainframe; 2nd generation computer	SISD
2	PDP-8	Scalar, sequential	Minicomputer using core memory	SISD
3	Intel 8080	Scalar, sequential	Microprocessor	SISD
4	IBM System/360 Model 30	Scalar, sequential	Mainframe; 3rd generation computer	SISD
5	IBM System/360 Model 91	Superscalar, pipelined	Mainframe; 3rd generation computer	SISD
6	Intel 80386	Scalar, pipelined	Microprocessor	SISD
7	Intel Pentium	Superscalar, two way; pipelined	Microprocessor with three execution units	SISD
8	Intel Pentium-PRO	Superscalar, three way; pipelined	Microprocessor: five execution units	SISD
9	Power PC	Superscalar	Microprocessor; RISC	SISD
10	CRAY-1	Vector processor	Supercomputer	SISD
11	TI ASC	Vector processor	Supercomputer	SISD
12	ILLIAC IV	Array processor	64 processors in a single system	SIMD
13	STARAN	Array processor	256 (bit-serial) processors	SIMD
14	FSP 164/MAX	Array processor	Attached array processor	SIMD
15	IBM 2938	Array processor	Attached array processor for IBM System/360 Models 44, 65 or 75	SIMD
16	Pluribus	Multiprocessor	Interface Message Processor (IMP)	MIMD
17	IBM System/370 Model 158 MP	Multiprocessor, CISC	Mainframe	MIMD
18	IBM System/360 Model 67	Multiprocessor, CISC	Designed to support efficient time sharing	MIMD

(Contd.)

Sl. no.	Processor/system	Types	Additional remarks	Flynn's type
19	Intel Pentium II	Superscalar with Multi-Media Extension (MMX), CISC	MMX operates on eight data elements simultaneously	SIMD MMX
20	CDC 6600	Superscalar, CISC	One central processor with 10 functional units	SIMD
21	UltraSPARC III	Superscalar, RISC	Supports multiprocessing	SISD
22	Intel 860	RISC microprocessor	Has on-chip graphics unit	SISD

12.7 Multiple Functional Units

Figure 12.5 presents a system with multiple hardware functional units operating in parallel inside a single processor. The instruction dispatch unit assigns the current instruction to the relevant unit based on the decoding result. The assigned unit continues with the subsequent steps, such as operand address calculation, operand fetch, execution etc. As soon as the current

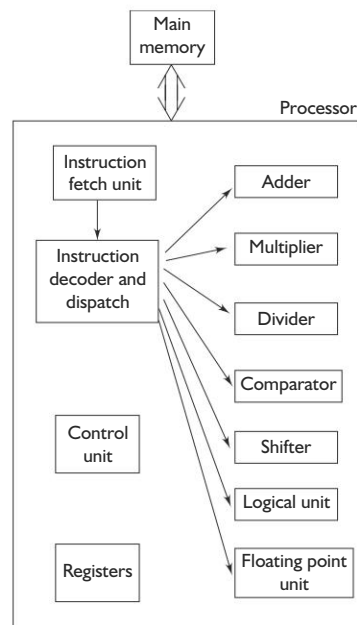


Fig. 12.5 Multiple functional units in a single processor

instruction is assigned to the functional unit, the instruction dispatch unit assigns next instruction to another unit, when it is free. It can have more than one functional unit of the same type such as multiple adders, multiple shifters etc. The control unit becomes more complex than the one in a simple processor. However, the throughput achieved is several times higher. The control unit should be alert in conflict handling such as dependency cases. For example, an instruction A whose operand depends on the execution of the instruction B, should not be executed until B is completed.

12.7.1 CASE STUDY 1—CDC 6600

The CDC 6600 is an example of a high performance computer with multiple functional units. The CDC 6600 is known as a network computer, since it has a special organization with one central processor, which consist of 10 functional units, and 10 peripheral and control processors. The 10 functional units within the central processor are as follows:

- One fixed point adder
- Two multipliers
- One divider
- Two incrementers
- One floating-point adder
- One shifter
- One logical unit
- One branch unit

The central processor has 24 useful registers:

- eight index registers
- eight operand address registers
- eight floating-point registers

The program does not see the functional units, and hence does not need any special modification. The control unit takes care of routing the instructions to functional units, and taking care of 'clash' situations. As long as there are no conflicts present in the instructions, up to 10 instructions can be issued in the central processor.

Figure 12.6 shows the functional units and registers of CDC 6600. The central processor consist of an instruction stack that can store up to eight words of 60 bits. The instructions are of two types: 16 bit and 30 bit instructions.

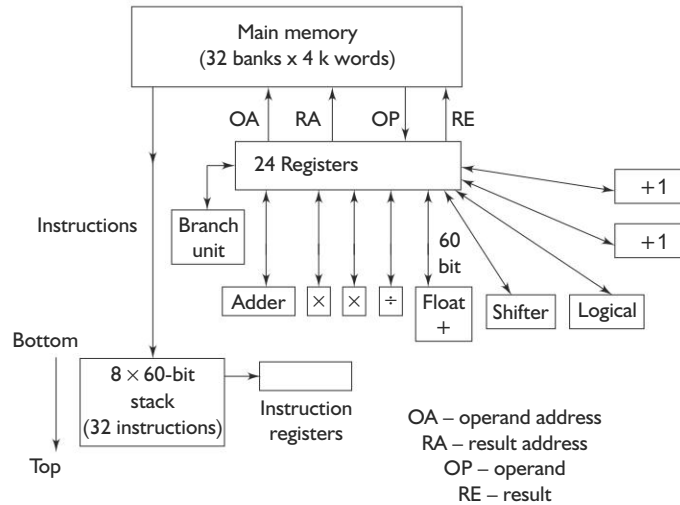


Fig. 12.6 CDC 6600 central processor—registers and functional units

The following are the different items of concurrency in CDC 6600:

1. Instruction prefetch: eight words of 60 bits each.
2. 10 functional units along with a 'scoreboard' that provides a queue and reservation scheme.
3. 32-way memory interleaving.
4. 10 peripheral processors with own memory for I/O programs.

12.8 Pipelining

Pipelining is a technique of splitting one task into multiple subtasks, and executing the subtasks of successive tasks parallelly, in multiple hardware units or sections. In a simple processor (scalar, non-pipelined), the steps of an instruction cycle are sequentially performed one after the other, and execution of successive instructions are also done sequentially, one after the other. Instruction pipelining is a technique, for instruction cycle implementation, in which execution of successive instructions are overlapped. The goal is increasing the total number of instructions executed in a given time period. The pipelining concept is similar to the assembly line in a production unit of a car factory. The assembly process of a car is split into small sub tasks. Each sub task is carried out at a specialized station/section. At a given instant, different stations will perform different subtasks on different cars. Similarly, in a pipelined processor, different sections of the processor perform different steps of the instruction cycle for different instructions at a given time. Each step is

called a pipe stage. All the pipe stages together form a pipe. Figure 12.7 shows the pipelining concept. The $S_1, S_2, \dots, S_n$ are n sections that form the pipeline. Each section performs a subtask on the input received from the interstage buffer that stores the output of the previous section. The interstage buffers isolate the adjacent sections. All sections can perform their operations simultaneously. The objective is to achieve the increased throughput due to the overlap between the execution of the successive tasks. The time taken for the execution of each task is same as the time taken for a sequential (non-pipelined) execution. But, due to concurrent execution of successive tasks, a number of tasks that can be executed within a given time is higher i.e., a pipelined hardware provides better throughput than a non-pipelined hardware.

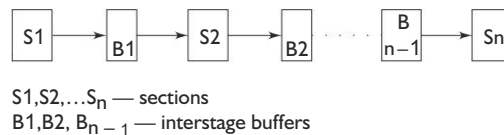


Fig. 12.7 Concept of pipelining

There are two types of pipelines:

1. Instruction pipeline
2. Arithmetic pipeline

An *Instruction pipeline* splits an instruction cycle actions into multiple steps that are executed one-by-one in different sections of the processor. An *Arithmetic pipeline* divides an arithmetic operation, such as a multiply, into multiple arithmetic steps each of which are executed one-by-one in different arithmetic sections in a pipelined ALU. A processor consist of either one or both types of pipelines.

The concept of an arithmetic pipeline is discussed for floating-point operations in Chapter 15. The following discussion deals with instruction pipeline.

12.8.1 Instruction Pipeline

An instruction pipeline consists of multiple sections each of which perform one of the phases of the instruction cycle. The processor hardware is structured into different independent sections for this. The number of sections (and the number of steps in the instruction cycle) in the pipeline is designed by the computer architect. The simplest case is a two-stage pipeline as shown in Fig. 12.8. The instruction cycle is split into two steps—Instruction Fetch (FI) and Instruction Execute (EI). The Fetch Unit (FU) performs instruction fetch step,

whereas the Execution Unit (EU) carries out the remaining actions of the instruction cycle. The Instruction Buffer (IB) temporarily stores the instruction fetched by the FU. The FU transfers the instruction into IB, and the EU receives the instruction from the IB. Figure 12.9 shows the timing diagram of the two-stage instruction pipeline while executing four instructions i.e. from I1 to I4. We assume that the time taken by FU or EU for its operation is just one clock cycle. Each instruction is completed in two clock cycles. Hence, the first instruction is completed at the end of second clock. The second instruction is completed at the end of third clock cycle. From the figure, it is observed that the number of clock cycles needed to execute four instructions is five. In a non-pipelined processor, eight clock cycles are needed to complete four instructions.

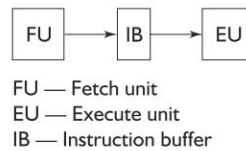


Fig. 12.8 Two-stage pipeline

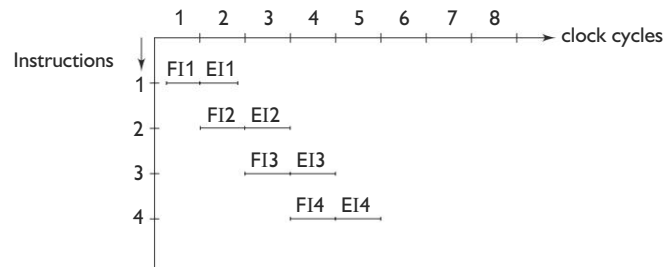


Fig. 12.9 Timing diagram for two-stage pipeline

Figure 12.10 exhibits a four-stage instruction pipeline, and Figure 12.11 shows the timing diagram for executing six instructions. Figure 12.12 presents a six-stage instruction pipeline.

The rate at which instructions come out of the pipe indicates the throughput of the pipeline. All the stages should be synchronized so that the various stages start their actions at the same time. Also, time required by different stages of a pipeline is different. Some stages finish the relevant steps faster, but they cannot start their steps for next instruction and

should wait till all the stages complete their steps since all the stages should operate in synchronization.

12.8.2 Speed-up

We have observed that a two-stage pipeline takes five clocks to complete 4 instructions, whereas a non pipelined processor requires 8 clock cycles (4×2). Similarly, a four-stage pipeline requires 9 clocks to complete six instructions, whereas a non pipelined processor requires 24 clocks (6×4).

The following analysis establishes that time taken to complete m instructions in a n -stage pipeline is approximately n .

Time taken for first instruction = nt_c where, t_c is the duration of one clock cycle.

Time taken for remaining $(m - 1)$ instructions = $(m - 1)t_c$.

Total time taken for m instructions = $(nt_c) + (m - 1)t_c = (n + m - 1)t_c$.

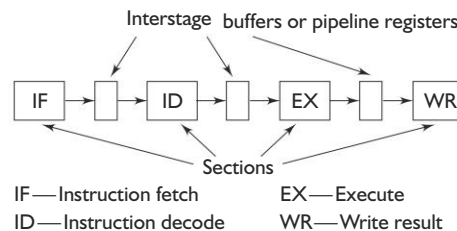


Fig. 12.10 A four-stage pipeline

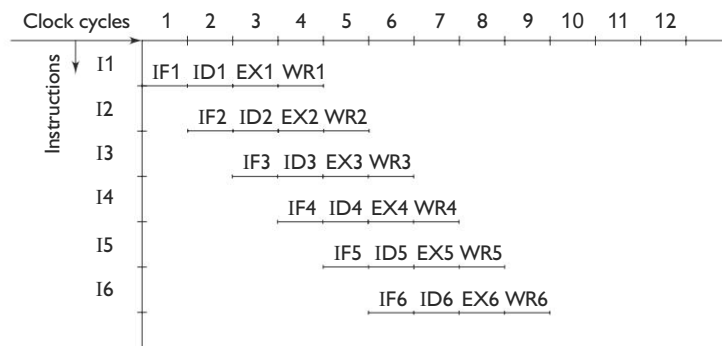
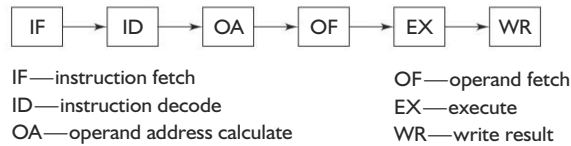


Fig. 12.11 Timing diagram for four-stage pipeline



Note : Interstage buffers are not shown

Fig. 12.12 A six-stage pipelined processor

If the processor is non-pipelined, the time taken for m instructions is nmt_c assuming the instruction cycle time is equal to nt_c .

Performance gain due to pipeline = time taken in non-pipelined mode/time taken in pipelined mode = $\frac{nmt_c}{(n+m-1)t_c} = \frac{nm}{n+m-1}$

For a large value of m , it is much larger than $n-1$. Hence $(n+m-1)$ approaches m .

Hence, performance gain or speed-up = $nm/m = n$.

Thus, the theoretical maximum speed-up is equal to the number of stages in the pipeline. Number of clock cycles needed for executing m instructions is m , since one instruction is completed per clock cycle. In practice, either the program logic or hardware reduce the speed-up gained due to pipelining, and increase the number of clock cycles needed for m instructions.

12.8.3 Pipeline Parameters

The end goal of pipelining is to increase the productivity—the number of instructions executed per second. In the processor pipeline, the execution of each instruction is divided into a sequence of simple sub-operations. Each sub-operation is performed by a separate stage. Each stage passes the result to the succeeding stage. Usually, an instruction remains in each stage only for a single clock cycle, and each stage begins to execute a new instruction as the previous one gets completed simultaneously in the later stages. Thus, a new instruction usually begins in every cycle.

Pipelines improve the rate at which instructions can be executed, as long as there are no dependencies. The efficient use of a pipeline requires several instructions to be executed in parallel. However, the result of an instruction is not available, for several cycles after it enters the pipeline. Therefore, the new instructions that depend on the results of instructions which are still in the pipeline, must wait.

12.8.3.1 Pipeline Latency

The latency of an execution pipeline is the number of clock cycles between the time of issuance of an instruction, and the time when a dependent instruction (which uses its result as an operand) can be issued.

12.8.3.2 Pipeline Repeat Rate

The repeat rate of the pipeline is the number of clock cycles that occur, between the issuance of one instruction and the issuance of the next instruction, to the same execution unit:

As discussed earlier, pipelines are used in two areas of computer design— instruction processing and arithmetic operations. The following requirements must be fulfilled in a pipelined system:

- A basic function must be divisible into independent stages that have minimal overlap.
- The complexity of the stages should be roughly similar.

The number of stages is referred to as the *depth* of the pipeline.

There are some instructions which need all the stages of the pipeline, whereas others require only a few stages. However, the instructions which require only some stages cannot bypass the unwanted ones. In other words, they cannot overtake the partially executed preceding instructions. As a result, the net speed-up of the pipelining is reduced.

12.8.3.3 Clock Frequency

The time taken for the instruction steps at various stages is different. For example, if the instruction fetch needs more time than instruction decode, the clock period of the pipeline should be not less than the time taken by the longest stage in the pipeline. As a result, those pipeline sections which complete their operations faster are forced to wait. To minimize this waiting time and achieve better efficiency from the pipelining, longer steps are split into more than one small stage, so that time required for each stage, is reduced. This permits a higher clock frequency. In addition, alternate solutions reduce the time taken for long activities.

Pipeline efficiency: Qualitatively speaking, the pipeline is not used to its maximum efficiency during initial filling time and end flushing time. The efficiency or utilization factor (E) of the pipeline is always less than 1. Hence, speed-up = nE .

Example 12.1 A six stage instruction pipeline has an utilization factor of 0.7. What is the speed-up due to pipelining?

$$\begin{aligned}\text{Speed-up} &= nE \\ &= 6 \times 0.7 = 4.2\end{aligned}$$

Example 12.2 A program takes 500 ns for execution on a non-pipelined processor. Suppose we need to run 100 programs of same type on a five stage pipelined processor with a clock period of 20 ns. What is the speed-up ratio of the pipeline? What is the maximum achievable speed-up?

Time taken by one program on non-pipelined processor = $mnt_c = 500$ ns.

Assuming the instruction cycle takes five clocks of each 20 ns, $m = mnt_c / nt_c = 500/100 = 5$. Hence, one program has five instructions. Number of instructions in 100 programs = 500 instructions.

Let us calculate the speed-up while running 100 programs in the pipelined processor.

Performance gain due to pipeline is = time taken in non-pipelined mode/time taken in

$$\text{pipelined mode} = \frac{mnt_c}{(n + m - 1)t_c}$$

Here, $n = 5, m = 500, t_c = 20$ ns

$$\begin{aligned} \text{Speed-up} &= (5 \times 500 \times 20) / (5 + 500 - 1) \times 20 \\ &= 50000 / (504 \times 20) \\ &= 4.96 \end{aligned}$$

Maximum speed-up = $n = 5$

In an ideal pipeline, the stages are properly balanced. In such a case, the time taken per instruction on pipelined processor is equal to the time taken per instruction in non-pipelined processor divided by the number of pipe stages. In a practical pipeline, the stages will not be perfectly balanced. Also, there is some overhead in the pipeline. Hence, the time per instruction on pipelined processor will be slightly more than the ideal one. The average execution time per instruction is reduced by pipelining. In other words, pipelining decreases the number of clock cycles per instruction (CPI).

12.8.3.4 Pipeline Performance

In a pipelined processor, the number of instructions completed in a given time is higher than that in a non pipelined processor. However, the execution time, for each instruction in the pipeline, is slightly more due to overheads and propagation delays. Thus, in a pipelined processor, every instruction runs relatively slower, but the program runs relatively faster due to overlapping of instructions. The following factors limit pipeline performance:

1. Pipeline latency
2. Imbalance among the pipe stages: The period of the clock should be more than the time taken by the slowest pipe stage.

3. Pipeline overhead: Two factors contribute to the pipeline overhead – pipeline register delay and clock skew. The pipeline register delay is caused by the set up time for the input and propagation delay to the clock signal.

12.8.3.5 Super Pipeline

A super pipeline uses a large number of stages usually more than 8 stages. A super pipeline operates at a higher clock frequency with each stage performing a small subtask. For example, the Pentium-pro processor has a 14-stage super pipeline. Its pipeline latency is shorter than that of Pentium processor's pipeline.

12.8.4 RISC Instructions and Pipelining

Though pipelining can be implemented in both CISC and RISC types of processors to enhance performance, it is simpler to design a pipelined RISC processor. The following properties of RISC architecture help in simplifying the pipeline design:

1. All instructions are of equal size, say 4 bytes.
2. Instruction formats are not many; just 1 to 3.
3. Arithmetic and other operations on data always have operands (data) in registers (not in memory).
4. Only load and store instructions can access memory.

Generally, RISC processors have three types of instructions: ALU instructions, Load and store instructions and Branch and Jump type instructions.

ALU Instructions

For these instructions, the operands are available in registers. On completion, the results should be stored in registers.

Load and Store Instructions

For these instructions, one operand is in register and the other operand is in memory. The address of the memory operand is generally specified as the sum of two parts: the base register contents and the offset indicated by the immediate field in the instruction.

Branches and Jumps

The branch conditions are usually specified in one of the two ways:

1. Comparison of two items in registers
2. Condition bits or condition codes

Unconditional jumps are present in almost all RISC processors.

12.8.4.1 Instruction Cycle Design for RISC Processors

Let us consider a RISC Processor without pipelining. Assume that the processor supports only three types of instructions: ALU instructions, Load and Store instructions, and Branches and Jumps. A close look at the actions performed in the instruction cycles of these three types gives us a clue that a five clock cycle sequence can be chosen for the instruction cycle as shown in Fig.12.13. The top portion of each phase names the instruction cycle phase and the corresponding bottom portion identifies the hardware resources needed. Table 12.5 defines the clock cycles, respective stages of instruction cycle and micro operations. Actual number of clock cycles required for different instructions are as follows:

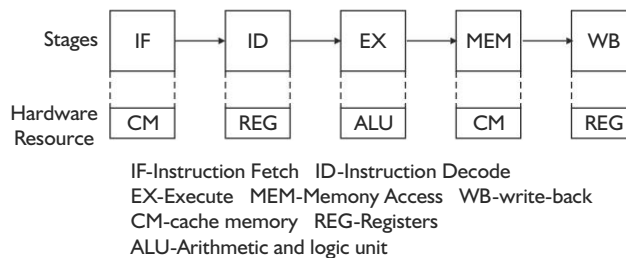


Fig. 12.13 Typical instruction cycle sequence in RISC processors

Unconditional branch instruction: 2 (cycles 1 and 2)

Store instruction: 4 (cycles 1 to 4)

Any other instruction: 5 (cycles 1 to 5)

There are many alternate design options offering varying performance levels. The designer chooses the best option taking into account hardware cost and required performance level.

TABLE 12.5 Typical instruction cycle phases in RISC processors

Sl. no.	Clock cycle	Instruction cycle phase	Major micro operations	Hardware sections involved
1	1	Instruction Fetch (IF)	a. Send PC contents to memory b. Fetch the current instruction from memory c. Increment PC by 4 to point to next instruction address	a. Cache memory
2	2	Instruction Decode (ID) for all instructions; plus Register Read cycle	a. Decode the instruction b. Read the contents of source registers c. Compare the contents of registers (as preparation for certain instructions)	a. Instruction decoder b. Registers c. Adder/comparator
3	3	Execution (EX); plus Effective address cycle	a. For ALU instruction, the specified operation is done by the ALU b. For memory reference instruction (Load/store), the effective address is calculated by ALU by adding the base register contents and the offset. c. For branch instruction, testing of branch condition is done.	a. ALU b. ALU c. ALU
4	4	Memory Access (MEM); plus branch completion	a. For load instruction, memory read operation from the effective address is done. b. For store instruction, memory write operation at the effective address, storing the contents of source register c. For branch instruction, the branch address is entered in PC if branch occurs	a. Cache memory b. Cache memory
5	5	Write - back (WB)	a. The result is stored in the destination register for load instruction and ALU instruction.	a. Registers

12.8.4.2 Five Stage Pipeline for RISC Processor

Figure 12.14 (a) shows the five stage pipeline for a typical RISC processor. Fig.12.14 (b) shows timing diagram while executing 6 instructions over 10 clock cycles. Fig.12.14 (c) shows the RISC pipeline as a series of data paths shifted in time. There are two major problems in a practical pipeline:

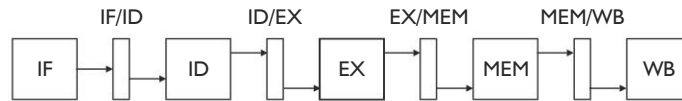


Fig. 12.14(a) A Typical five stage pipeline for RISC processors

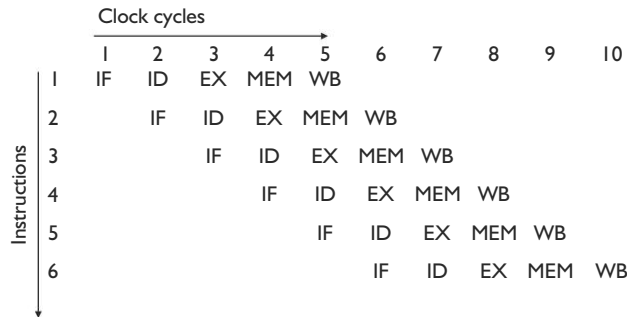


Fig. 12.14(b) Timing diagram—Instruction execution in RISC pipeline

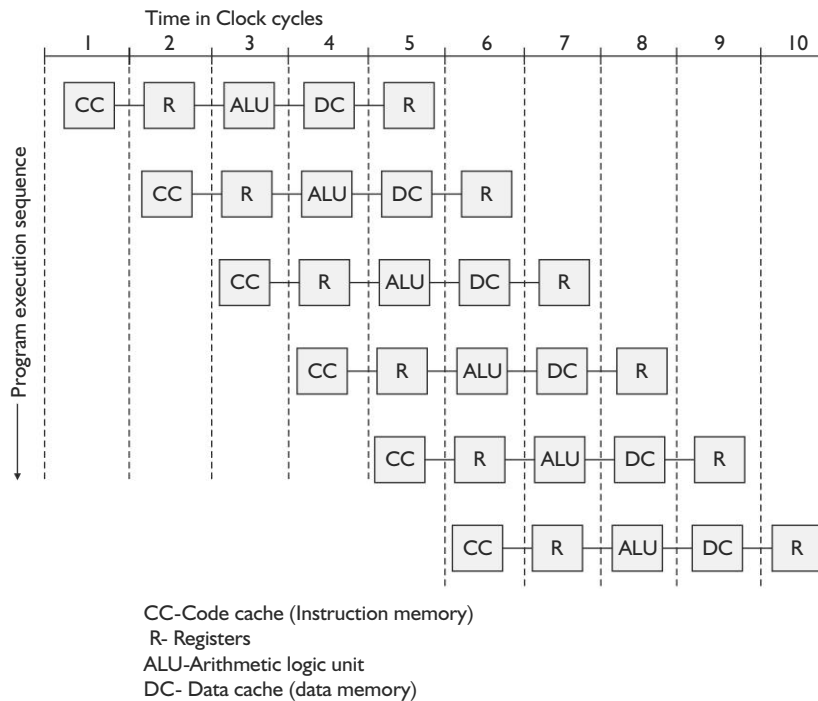


Fig. 12.14(c) RISC pipeline viewed as data paths shifted in time

1. Resource Conflict: Two different operations at two sections/stages may need the *same* hardware resource in the *same* clock cycle, due to overlapping of instructions. To resolve this, multiple resources of the same type can be provided in the hardware. This will increase the cost and hence should be done judiciously.
2. Interference between adjacent stages: Two instructions in different stages of the pipeline should not interfere with each other. To resolve this, we use pipeline registers between successive stages of the pipeline. The result of a given stage is stored in the pipeline register at the end of a clock cycle. During the next clock cycle, the contents of the pipeline register serves as input to the next stage. In some cases, the result generated by one stage may not be used as input to the next stage. It may *propagate* through more than one stage. For example, for a STORE instruction, the result is produced in the ID stage but it is stored in memory only in the MEM stage. The pipeline registers are named indicating the stages linked by them such as IF/ID, ID/EX, EX/MEM and MEM/WB.

12.9 Pipeline Hazards

The pipeline operates continuously as long as the program does not encounter any hazard situation. A hazard is a condition or situation in the pipeline due to which instruction processing can not continue as usual in the predetermined clock cycle, and the actions by some stages have to be stalled for some time, till the hazard disappears. In other words, a hazard is created if there is dependence between instructions in the pipeline. The cause for the hazard can be either the *hardware* or the *software*. The hardware cause for hazard is the shortage of resources in meeting simultaneous requirement of these, by various stages in pipeline, for different instructions in the pipeline. This can be reduced by some extent if sufficient resources are added, though at increased cost. On the other hand, the software cause is the property of program, being executed in the pipeline. A practical program has different types of inter-instruction relationships that result in stalling the pipeline for some-time. In other words, dependencies between successive instructions cause hazards in pipeline, and affect the smooth operation of the instruction pipeline. The three major causes for the hazards are as follows:

1. Structural Hazard or Resource Conflict
2. Data Hazard or Data Dependency
3. Control Hazard or Branch Difficulty

12.9.1 Structural Hazard

When two different sections in the pipeline need the same hardware resource simultaneously, it results in the resource conflict. Some typical examples are as follows:

1. Suppose the final stage WB needs register access for storing the result of a completed instruction, and at the same time, the ID also requires register access for fetching the operand, for the subsequent instruction. Obviously, one of them will be delayed, which will affect the pipeline performance. If we delay the WB's register access, it will stall the pipeline operation for subsequent instructions, till WB completes the register access. On the other hand, if we delay the ID's register access, partial operation of the instructions between ID and WB will proceed, whereas the IF section preceding ID will be frozen. Let us assume that the control unit gives priority to the WB stage and blocks the ID stage from performing its role. The effect of this on the pipeline flow is shown in Fig.12.15. The pipeline is stalled for one clock cycle when the WB stage performs register write operation. As a result, no new instruction can enter the pipeline. During this clock cycle, the MEM and EX stages are allowed to function, but IF and ID stages are disabled. Pipeline stall is known as *pipeline bubble* which will be discussed shortly. Alternate solution to this resource conflict, without stalling the pipeline in the above situation, is designing the registers with dual port access. In this case, simultaneously the WB and ID stages can access the registers, and pipeline is not stalled.

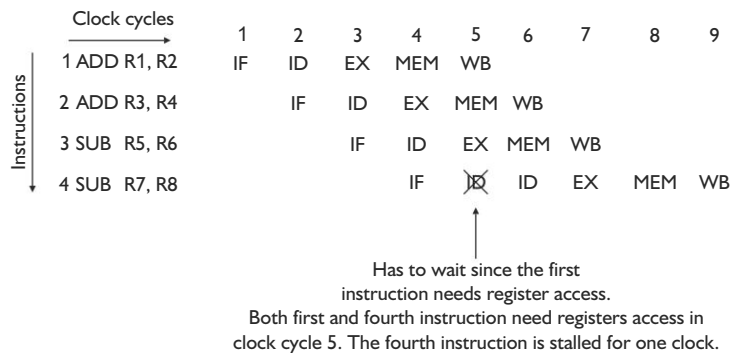


Fig. 12.15 Resource conflict due to simultaneous Register access by ID and WB stages

2. Another type of resource conflict occurs when the IF stage requires memory access for the instruction fetch, and MEM stage needs memory access for the operand fetch at the same time. Figure 12.16 illustrates resource conflict when instruction 1 (LOAD)

wants to fetch the operand from memory in clock 4, and at the same time, instruction 4 wants to fetch an instruction from memory. Hence, the IF stage is stalled for one clock. Frequent stalling of pipeline reduces the effective performance gain due to pipelining. This can be resolved by having separate cache memory units for the instructions (code cache) and operands (data cache), as in the case of Pentium microprocessor. Simultaneously, both code cache and data cache can be accessed. This concept is known as split cache. In case there is unified (common) cache instead of split cache, pipeline has to be stalled temporarily. However, even with split cache, stalling of pipeline may be required depending on the type of data hazard encountered in the instruction stream.

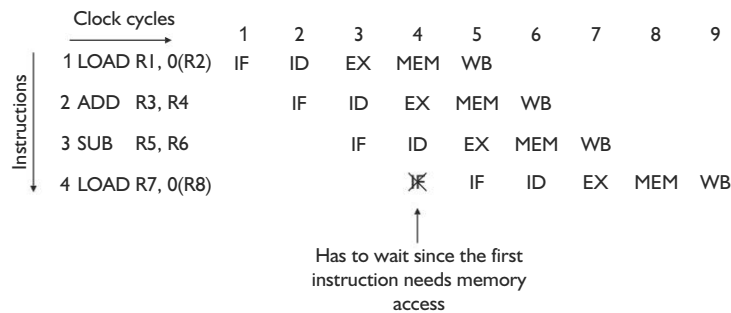


Fig. 12.16 Resource conflict due to simultaneous memory access by IF and MEM stages.

12.9.1.1 Pipeline Bubbles

When progress in pipeline is stalled due to disabling a stage from its operation, because of a hazard, some stages idle for one or more clock cycles till the hazard condition vanishes. These idle periods are called as bubbles or stalls. The bubbles travel through the pipeline and disappear soon. The time taken (in number of clock cycles), for the bubble to vanish, depends on the pipeline design.

12.9.1.2 Dependencies

Dependencies between instructions is a property of programs. If two instructions are dependent, they should not be executed simultaneously. They may be partially overlapped. Two instructions may be either directly data dependent, or indirectly data dependent, through another instruction, due to chain of dependencies. In case of dependence, there are two possible solutions:

1. Preserving the dependence but preventing a hazard
2. Removing the dependence by transforming the object code.

Techniques used for detecting and preventing hazards should preserve program order so that the overall behavior and results of the program are not affected.

12.9.2 Data Hazard

If the operand, for an instruction, is the result of the previous instruction, which is not yet completed in the pipeline, it is known as data dependency. Figure 12.17 shows pipeline stalling due to the non-availability of the operand, for the second instruction, since the first instruction is not yet completed. Nothing is done in clock cycles 4 and 5 for second and subsequent instructions. Hence, *penalty* has to be paid, due to data dependency, on the two clock cycles. A standard solution to the data dependency is known as ‘operand forwarding’. Figures 12.18(a) to 12.18(c) illustrate this concept. The result of Execute (EX) section is the feedback to the input of the same section, so that it is easily available (before the required time), for use in the subsequent steps, for the next instruction. The control unit enables this path at appropriate time.

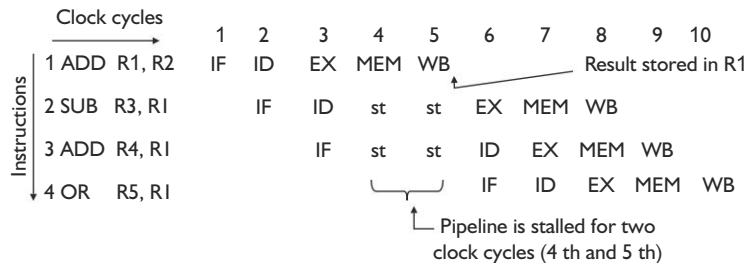
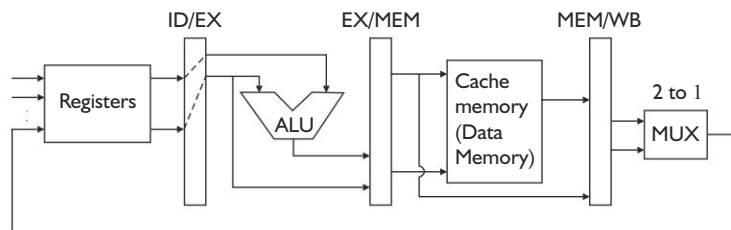


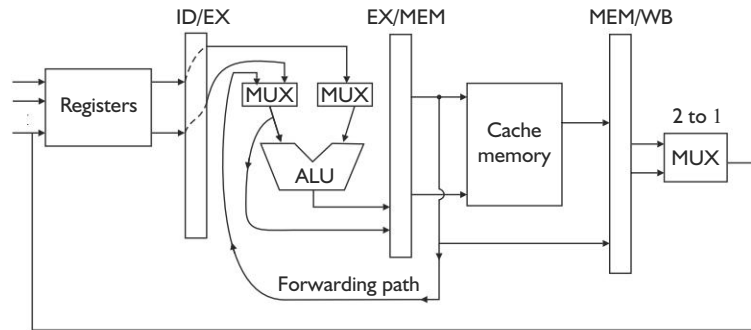
Fig. 12.17 Stalling due to data dependency—five stage RISC pipeline



1. MUX control not shown. 2. Only partial pipeline shown. 3. Controls are not shown.

Fig. 12.18(a) Simplified data path (without operand forwarding)

The data hazards are classified into three types depending on the order (sequence) of read and write accesses made by the instructions in the pipeline. Let us consider two instructions A and B, with A preceding B in the program but not necessarily adjacent to B. The three types of data hazards are: Read-After-Write (RAW) hazard, Write-After-Write (WAW) hazard and Write-After-Read (WAR) hazard.



1. Only one forwarding path shown
2. Controls are not shown.

Fig. 12.18(b) Partial pipeline data path with operand forwarding

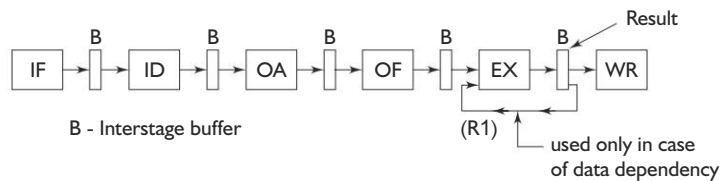


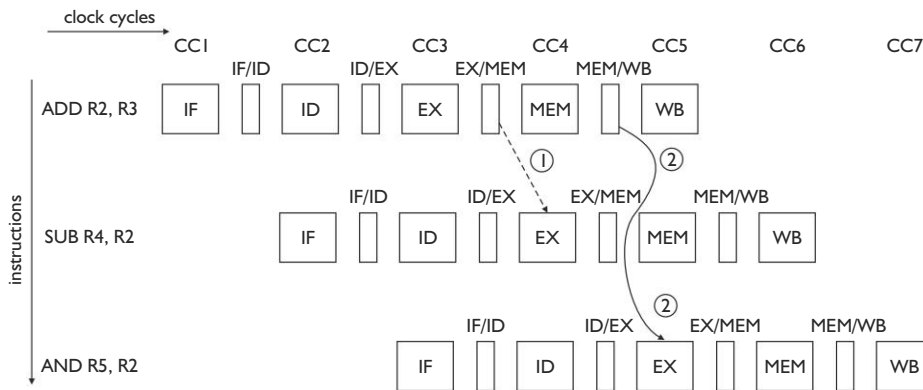
Fig. 12.18(c) Operand forwarding-six stage pipeline

RAW Hazard: This situation occurs if B needs an operand that is stored by A. Due to overlapping of A and B, B may try to read the operand from the source, before A has stored it. If this is allowed, B will read a wrong (old) value. Consider the pipelined execution for the following program segment:

```
ADD  R2, R3
SUB  R4, R2
AND  R5, R2
```

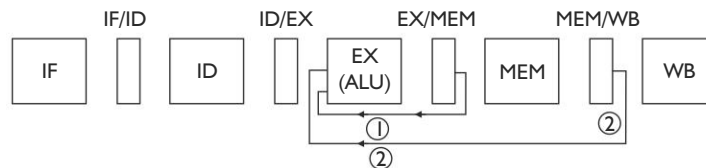
The ADD instruction will store the result in R2 in the last stage, WB, in clock cycle 5 (at the end), though the result is produced in clock cycle 3 by the ALU, in EX stage. But, the SUB instruction will read the operand from R2 in second stage, i.e. in clock cycle 3, for

performing subtraction in clock cycle 4. Hence, RAW hazard exists between these two instructions. As a simple solution, the SUB instruction should not be allowed to read R2 value, until the beginning of clock cycle 6. Hence, the ID and IF stages should be stalled during clock cycles 3 to 5. Alternatively, the result can be forwarded to ALU in clock cycle 4 from the pipeline register, EX/MEM. This eliminates the stalling. The third instruction also needs the same value of R2 for AND operation in clock cycle 5 (in the beginning). This also can be met by a forwarding path, from the pipeline register MM/WB to the ALU, in clock cycle 5. The pipeline control should be carefully designed to enable appropriate forwarding paths at the right clock cycles, depending on the instruction sequence, and the need for the operands based on their dependencies. Figures 12.19 (a) and (b) show the forwarding paths used for eliminating RAW data hazard.



The path 1 should be enabled in CC4 only. The path 2 should be enabled in CC5 only.

Fig. 12.19(a) RAW hazard and operand forwarding—timing diagram



① is used in CC4 for SUB instruction. ② is used in CC5 for AND instruction.
① forwards from immediately previous instruction. ② forwards from an instruction commenced two clocks earlier.

Fig. 12.19(b) RAW hazard and operand forwarding—Hardware paths

WAW Hazard: This situation occurs if both A and B store their operands in the same destination. As per program order, B should write in the destination, only after A completes writing in it. Due to overlapping of A and B, B may try to write the operand in the destination before A writes in it. If this is allowed, any other instruction, preceding B and requiring the same operand, will read a wrong (new) value. Consider the pipelined execution for the following program segment:

```
LOAD    R2, 0(R3)
STORE   0(R4), R2
SUB      R2, R4
```

Both the LOAD instruction and SUB instruction write in R2 and the STORE instruction reads from R2. The SUB instruction should not modify R2, before store instructions reads R2.

WAR Hazard: This situation occurs if A reads an operand from the source which is used by B as destination. Due to overlapping of A and B, B may try to write in the destination, before A reads from it for the source operand. If this is allowed, A will read a wrong (new) value. Consider the pipelined execution for the following program segment:

```
STORE 0(R4), R2
SUB R2, R5
```

The SUB instruction writes in R2 and the STORE instruction reads from R2. The SUB instruction should not modify R2, before store instructions reads R2. The early detection of the hazardous situations, and commencement of the correctives to prevent the hazards is usually carried out by the appropriate hardware logic. A superscalar processor has extensive hardware circuits for this purpose, as discussed in Chapter 13.

There are also some instruction combinations causing data hazard in pipeline for which pipeline stalling is unavoidable, since no solution to avoid stalling is possible. Consider following program segment:

```
LOAD  R2, 0(R4)
ADD   R3, R2
SUB   R6, R2
```

The value to be loaded in R2 is read from memory, by the LOAD instruction, in clock cycle 4 in MEM stage, and stored in R2 in the last stage, WB, in clock cycle 5. But, the ADD instruction needs this value in EX stage, in the beginning of clock cycle 4. Hence, forwarding is not possible. Only option available is to stall the pipeline for one clock cycle. Figures 12.20(a) and 12.20(b) show the timing diagrams for both cases: (a) no hazard detection case and (b) hazard detection and stalling case.

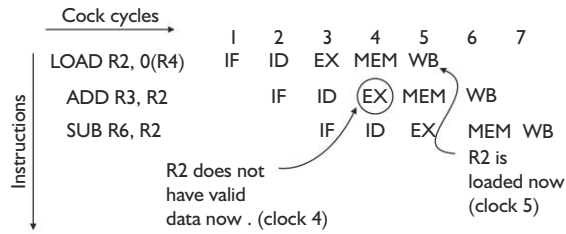


Fig. 12.20(a) Program execution without detecting data hazard

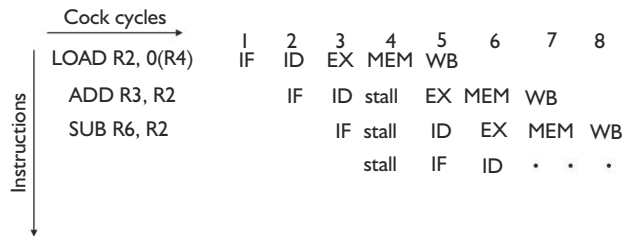


Fig. 12.20(b) Data hazard detection and pipeline stalling

Software Solution to Data Hazard

To avoid data hazard in pipelining, as a software solution, the compiler senses the hazard by scanning the instructions in the program. It looks for some unrelated instructions in the program, that can be shifted, and placed between the two instructions (source and dependent) that have data dependency. However, the compiler should ensure that program logic is not affected due to such movement of instructions. To eliminate the pipeline stall, the compiler should separate the two dependent instructions, by number of clock cycles equal to the pipeline latency of the first instruction (source). In case the compiler is unable to find any unrelated instruction in the program, it can insert one or more NOOP instructions between the two dependent instructions. As a result, a delay of two or more clock cycles occurs and resolves the data dependency. The merits of this method are simpler hardware and freedom to insert some useful instructions, in the NOOP slots, wherever possible by the compiler. However, the program size increases with the introduction of the NOOP instructions. The concept of reordering the object code by the compiler is known as static scheduling. This is followed by some superscalar processors (Chapter 13) and VLIW processors (Chapter 14).

12.9.3 Instruction Hazard (Control Hazard)

Control hazards (Instruction hazards) are caused by branch instructions, both unconditional and conditional branches. Unconditional branch instructions cause branching always, whereas conditional branch instructions may or may not cause branching. In a pipelined processor, following actions are critical in handling control hazard:

1. Timely detection of a branch instruction
2. Early calculation of branch address
3. Early testing of branch condition (fate) for conditional branch instructions

Different processors follow different techniques to minimize pipeline stalling duration. When a branch instruction enters the pipeline, it is recognized during the instruction decode step. Different action is performed by the processor for unconditional and conditional branch instruction, as shown in Fig 12.21 for a non-optimized processor. Figure 12.22 illustrates the

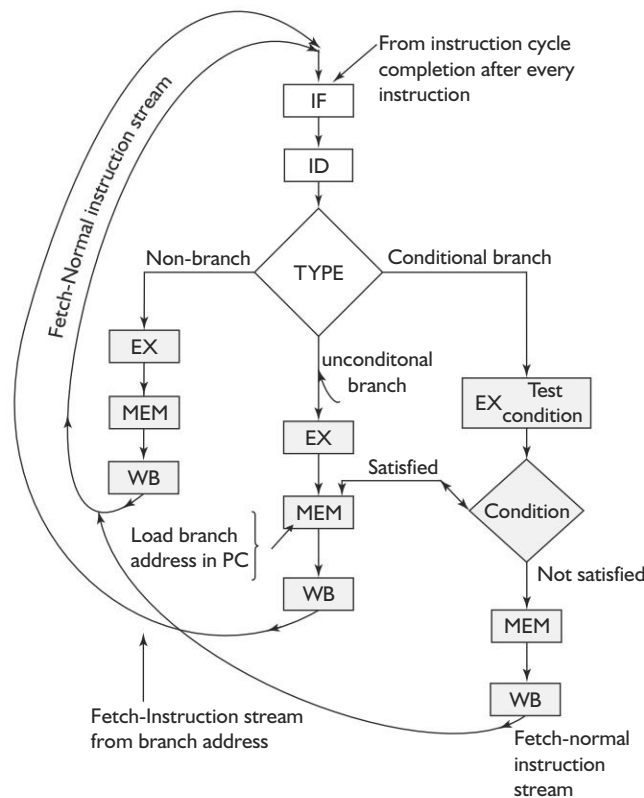


Fig. 12.21 Processor actions and branch instructions

actions in a five stage pipeline for an unconditional branch instruction. The processor decodes the 'branch' in ID stage (in clock 3 in Fig.12.22), and loads the branch address in PC in EX stage, in clock cycle 4. Hence the instruction from the target address can be fetched in clock 5. We have two clocks penalty due to this instruction. In clock 4, branch address is loaded in PC. Hence, the steps carried out in clocks 3 and 4 for third and fourth instructions are flushed. In clock 5, new instruction is fetched from the branch address. The unconditional branch instruction results in the actions IF3, ID3 and IF4 which are completed but not utilized.

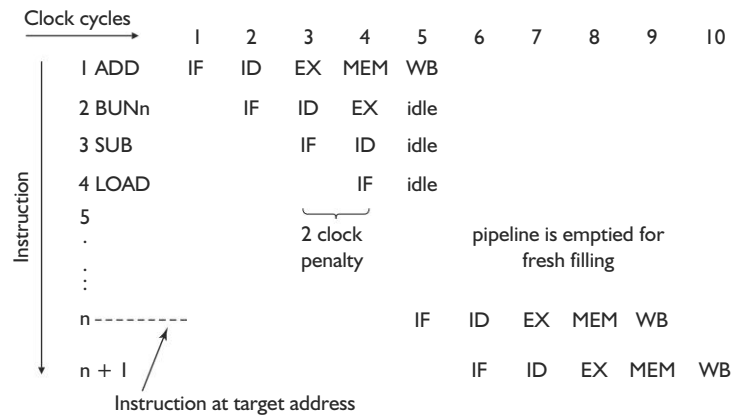


Fig. 12.22 Effect of branch penalty—unconditional branch

The branch instruction results in aborting of the partially executed instructions by flushing them out of the pipeline and a fresh refill of the pipeline. The presence of many branch instructions in the program reduces the performance gain (speed-up) achievable by instruction pipelining. If p is the fraction of number of branch instructions to total number of instructions, then $1/p$ is the limiting factor of speedup due to the branch instructions.

To minimize the branch penalty, the pipeline control is designed to take special steps. In an optimized processor, by using additional hardware logic, it is possible to complete the branch address calculation and loading it in PC in the ID stage itself.

A special action is performed by the hardware to sense the branch instructions in advance, when it is present in the instruction queue. In this case, the instruction in the branch address is fetched earlier than usual time. Figure 12.23 illustrates the effect of this on unconditional branch instruction. The branch address is loaded in PC in clock 2 whereas in clock 3, instruction fetch from this address is done. The branch penalty is reduced to just one clock cycle.

12.9.3.1 Delay Caused Due to Conditional Branch Instruction

A conditional branch instruction can create one of the two situations:

1. The condition tested is successful; here, the contents of the pipeline (instructions following the branch instruction that have been partly executed) are flushed out, and the instruction at the branch address (and following instructions) enter the pipeline.
2. The condition tested is unsuccessful, and hence there is sequential execution; in this case, the pipeline contents are not affected.

The delay caused by conditional branch instruction is well-explained in the following example.

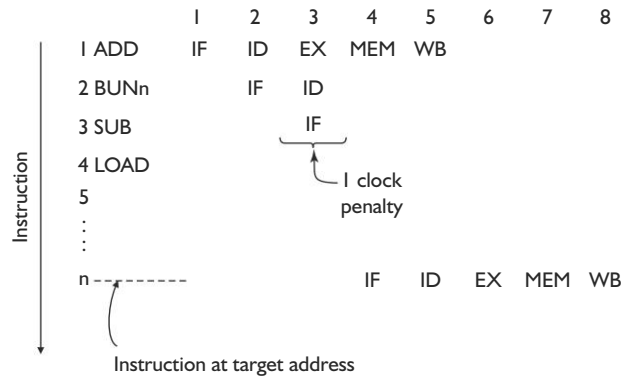


Fig. 12.23 Effect of branch penalty—use of additional logic

Suppose there are m instructions to be executed. Let p be the probability of the conditional branch instruction and q be the probability for the success of the branch. Hence, number of instructions causing successful branch is mpq (Figure.12.24).

In a pipelined processor, the instructions are completed at the rate of one instruction per clock cycle. If there are no branch instruction in the stream, the program routine is completed in m clocks. Each branch instruction results in flushing the pipeline and refilling from the branch address. The instruction from the branch address (target address) needs n clock cycles for execution.

Total clock cycles needed to execute the m instructions = (time delay in clock cycles introduced by branch instructions) + (clock cycles needed for non-branch instructions)

$$\begin{aligned}
 &= mpqn + (m - mp) + mp - mpq \\
 &= mpqn + (m - mpq) \\
 &= mpqn + m(1 - pq)
 \end{aligned}$$

Thus, m instructions are executed in $mpqn + m(1 - pq)$ clock cycles.

Average number of Clocks Per Instruction (CPI) = $1 + pq(n - 1)$

If the program causes no branching, $q = 0$; CPI = 1.

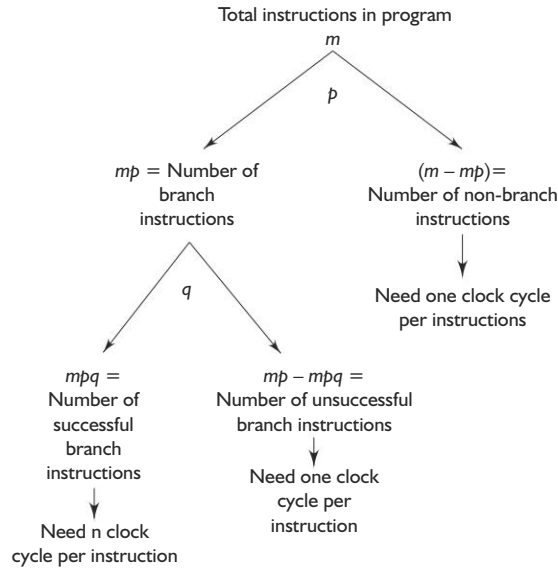


Fig. 12.24 Addition time involved due to branch

Example 12.3 A program has 20% branch instructions. Assuming all the branches are successful, calculate the speed-up, in a pipelined processor with six stages.

If there are no branch instructions in the program, the speed-up = n = no. of stages = 6

$$p = 0.2, q = 1.0, n = 6$$

$$\text{CPI} = 1 + pq(n - 1) = 1 + (0.2 \times 1.0 \times 5) = 2.0$$

A non-pipelined processor needs 6 clocks per instruction.

$$\text{Speed-up} = 6/2 = 3.0$$

Example 12.4 A program has 20% branch instructions. When it is run on a five stage pipelined processor, it was observed that 60% of the branches were successful. Calculate performance improvement in the pipelined processor. What is the maximum speed-up possible if there are no branches?

$$p = 0.2, q = 0.6, n = 5$$

$$\text{CPI} = 1 + pq(n - 1) = 1 + (0.2 \times 0.6 \times 4) = 1.48$$

A non-pipelined processor needs 5 clocks per instruction.

$$\text{Speed-up} = 5/1.48 = 3.38$$

Maximum speed-up in no-branch case = $n = 5$.

12.9.3.2 Minimizing Impact of Conditional Branch Instructions

Several techniques are followed in different processors for minimizing the delay due to conditional branch instructions (Fig. 12.25). The *delayed branch* is a software technique which is discussed in the following paragraph, whereas others are hardware techniques discussed in Chapter 13. In branch prediction, an advance (early) assumption is made regarding the branch as either 'branch will occur' or 'branch will not occur'. Based on this assumption (prediction), relevant actions will take place in pipeline for appropriate instructions, instead of simple freeze and stalling. A 'predicated branch untaken' approach is commonly followed one. In this, the processor assumes that (every) branch will not occur, and continues the pipeline actions as if there is no branch instruction. Once the condition testing is over, a final decision is taken depending on whether the assumption has proved right or wrong. In case of correct assumption, there is no branch penalty as shown in Fig. 12.26 (a). On the other hand, if the assumption proves to be wrong, a penalty of one clock cycle is present as shown in Fig. 12.26(b). Some processors follow the approach of 'predicted branch taken'. In this, the processor assumes that (every) branch will occur. Hence, it dumps the actions already taken for next instruction(s) and starts fetching instructions from the branch (target) address. On knowing the fate, corrective action will be done if the prediction turns out to be false. It is interesting to note the approach of INTEL 80486 processor; partly 'branch not taken' and partly 'branch taken'. This is discussed shortly. The superscalar processors follow advanced techniques, since they have multiple pipelines/functional units idling of which result in under utilization of hardware.

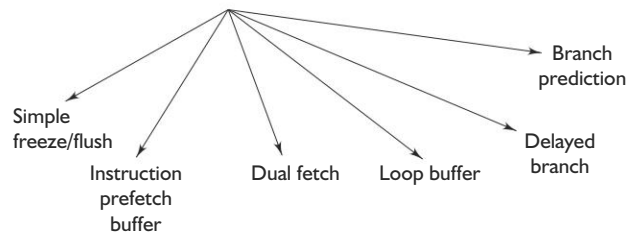


Fig. 12.25 Branch handling techniques

12.9.3.3 Delayed Branch

The instruction slot physically following the branch instruction is known as a delay slot. Accordingly, there are two delay slots in Fig. 12.22, and one delay slot in Fig. 12.23. The compiler tries to rearrange the instructions wherever possible (without affecting the program

logic) so that some unrelated instructions are moved after the conditional branch instruction, that can be continued in the pipeline. If the situation is such that no other instruction can be brought after the branch instruction, then the compiler inserts a NOOP instruction after the branch instruction. Figure 12.27 illustrates both these cases. The instruction in the delay slot is executed whether or not the branch is taken. If the branch is 'taken', execution continues at the branch target. If the branch is 'untaken', execution continues with the instruction following the branch delay instruction.

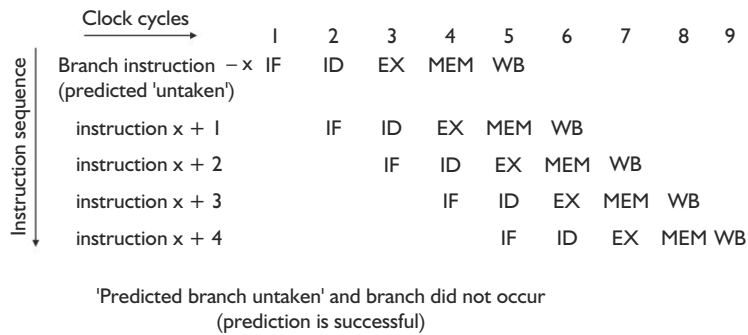


Fig. 12.26(a) Branch prediction - no penalty case

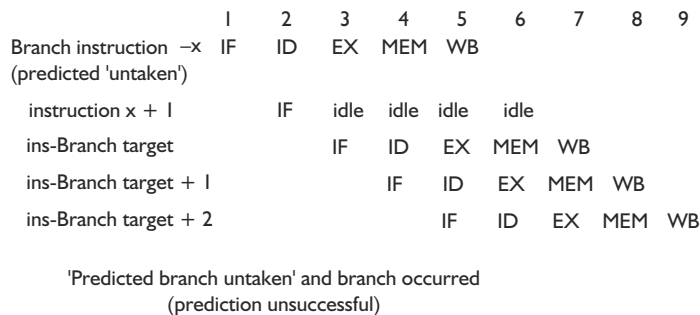


Fig. 12.26(b) Branch prediction - penalty case

12.9.3.4 CASE STUDY 2: Conditional Branch Handling in Intel 80486

The pipelined CPU takes special care in handling a conditional branch instruction. After decoding a conditional branch instruction, there are two possible options of design implementation for the pipeline control:

1. Continue fetch and decoding of subsequent instructions (assuming no branch) until the result of condition testing is known.
2. Fetching the instruction from the branch destination address (expecting a branch).

Depending on the results of the condition testing, dropping of the already fetched instructions and restarting a new instruction fetch may be necessary in both cases. This leads to lot of rework for the DO loop like situations in the program. To minimize this, Intel 80486 and Pentium follow various different strategies.

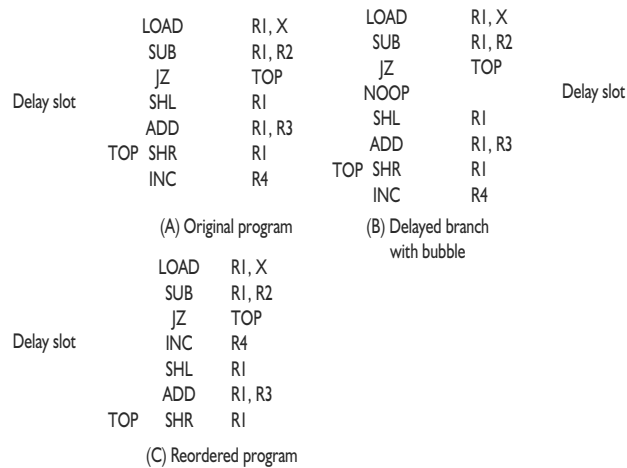


Fig. 11.27 Delayed branch by compiler optimization

12.9.3.5 INTEL 80486 and Conditional Branch

The pipeline stages of 80486 processor are displayed in Fig. 12.28. The 80486 follows the *cat-on-the-wall* strategy for conditional branch instruction. When a branch instruction is decoded, the pipeline continues as if a non-branch instruction is decoded. But, when the branch instruction reaches the EX stage, the pipeline fetches the instruction from the branch address. The fate (to branch or not to) is known only at the end of EX stage when the condition testing is carried out. However, in both the cases, 80486 is in a comfortable situation:

1. If branch does not occur, the prefetched instruction from the branch address is thrown out. The pipeline proceeds without any penalty (delay).
2. If branch occurs, the pipeline starts decoding the prefetched instruction from the branch address, and throws out the sequential instructions following the branch instruction. This situation has two clocks delay.

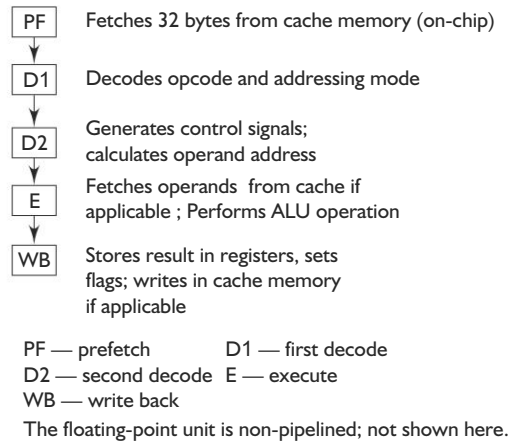


Fig. 12.28 Instruction pipeline in 80486

12.10 Influence on Instruction Sets

Execution of an instruction in a pipelined processor differs from execution in a non pipelined processor. Most of the instructions execute smoothly in pipeline without reducing the pipeline performance. But, some instructions either reduce pipeline performance or cause side effects. Since all modern processors are pipelined, impact of various instructions in pipeline execution should be taken into consideration while finalizing the instruction sets of processors.

12.10.1 Addressing modes

Complex addressing modes are avoided in pipelined processors. Instructions using complex addressing modes take long execution time and likely to cause pipeline stall. A processor designer prefers to have following features in the addressing modes:

1. Operand fetch should not require more than one memory access.
2. No side effects should be caused
3. Instructions other than load and store should not access memory for operand fetch or result store

These features are present in three basic addressing modes: register addressing, register indirect addressing and index addressing. They do not cause side effects. The index addressing mode involves operand address calculation in one clock cycle, and the operand

fetch in the next clock cycle. The other two addressing modes do not involve operand address calculation.

12.10.2 Condition Codes

While a compiler tries to reorder the instructions to avoid pipeline stall, it should not change the program logic due to instruction reordering. The condition codes set (or reset) by certain instructions should be taken into account by the compiler, so as not to disturb the program logic due to instructions reordering. To simplify the compiler's task, the instruction set designer should choose only few instructions that affect condition codes. In addition, the compiler should have provision to indicate the instructions which have affected the condition codes.

12.11 Data Path and Control Considerations

The data path needs suitable modifications for implementing a pipeline in the processor. An instruction is active at any instant only in one of the stages. All the stages in the pipeline should be able to act independently and simultaneously for different instructions. The data path design should have buffer registers, and also have duplication of certain registers to achieve the above requirement.

12.12 Exception Handling

Exceptional situations during program execution generally require temporary branching and returning back, without loss of CPU status or program logic. There are multiple causes for exceptional situations;

1. Program related unintentional causes such as overflow, illegal opcode, misaligned memory access, memory protection etc.
2. I/O controller/device related causes such as data transfer or status transfer
3. Program related intentional causes such as breakpoint, instruction trace, OS service request etc.
4. Hardware malfunction and power fail
5. Virtual memory interrupt (page fault)

Handling the exceptions is challenging for a pipelined processor in view of overlapped execution of instructions in the pipeline. Certain interrupts such as overflow, page fault, hardware malfunction may occur *within* instructions, whereas certain other interrupts such as I/O interrupt, breakpoint interrupt occur *between* instructions. Certain interrupts such as hardware malfunction and power failure cause *termination* of program execution. Certain other interrupts such as I/O interrupt, overflow, page fault and breakpoint enable resuming program execution.

When an exception occurs, the pipeline control should save the pipeline state, so that resuming the interrupted program is possible. In many systems, precise exception is a requirement. On returning from interrupt, the interrupted instruction should be completed, and other instructions in the pipeline after the interrupted instruction should be restarted from scratch. If such a restart is possible in the pipeline, we say the pipeline has precise exceptions. To achieve this, special actions by the hardware and/or software are required both on sensing an interrupt, and on returning from interrupt.

SUMMARY

The early computer designs followed the CISC approach that minimized the main memory size. The CISC processors supported a large instruction set including all the possible addressing modes. Though it increased the CPU complexity, it provided flexibility to the compiler/programmer. After the development of a low cost RAM chip, the computer designers concentrated on the processor performance and invented the RISC architecture. RISC processors have a limited number of simple instructions. Though it results in longer programs, the instruction execution is easier and faster. The processors have very little hardware and provides better performance than the CISC processors. Instruction pipelining and on-chip cache memory helped the RISC processor to complete one instruction for every clock cycle.

There are two basic strategies which are followed to enhance the performance of a computer:

1. *Overlap*: Splitting a task into multiple subtasks that can be performed in an overlapped manner.
2. *Parallelism*: Executing more than one task in parallel.

In *instruction level parallelism*, parallelism is applied in a single processor whereas in *processor level parallelism*, multiple processors share the workload (tasks). This can be achieved either by using more than one processor in a single computer system or by distributing the tasks among the multiple computer systems linked either as a cluster or a network.

Flynn's classification of computer organization defines four different types as follows:

Single Instruction stream, Single Data stream (SISD)

Single Instruction stream, Multiple Data stream (SIMD)

Multiple Instruction stream, Single Data stream (MISD)

Multiple Instruction stream, Multiple Data stream (MIMD)

A *simple scalar processor* performs only one arithmetic operation at a time. It executes one instruction at a given time and each instruction has one set of operands. In a *vector processor*, each instruction has multiple sets of operands such as elements of two arrays, and the same operation is performed simultaneously on different sets of operands. In a *non-pipelined (sequential) scalar processor*, there is no overlap of successive instruction cycles. In a *pipelined processor*, multiple instructions operate at various stages of the instruction cycle simultaneously, in different sections of the processor. An *array processor* has multiple execution units that operate simultaneously on multiple elements of a vector. A *superscalar processor* is a scalar processor that performs multiple instruction cycles simultaneously, for successive instructions.

The pipelining is a technique of splitting one task into multiple subtasks and executing the subtasks of the successive tasks in parallel. An instruction pipeline splits an instruction cycle actions into multiple steps that are executed one-by-one in different sections of the processor. An arithmetic pipeline splits an arithmetic operation into multiple arithmetic steps each of which are executed one-by-one in different arithmetic sections of the ALU. The branch instructions in the program reduce the speedup gained due to pipelining, and increase the number of clock cycles needed for m instructions. The dependencies between the successive instructions cause hazards in the pipeline, and affect smooth operation of the instruction pipeline. The following are three major causes of hazard:

1. Structural Hazard or Resource conflicts
2. Data Hazard or Data dependency
3. Control Hazard or Branch difficulty

REVIEW QUESTIONS

1. The data channel handles I/O operations by executing I/O commands, as discussed in Chapter 9. Discuss the merits and demerits of pipelining the data channel.

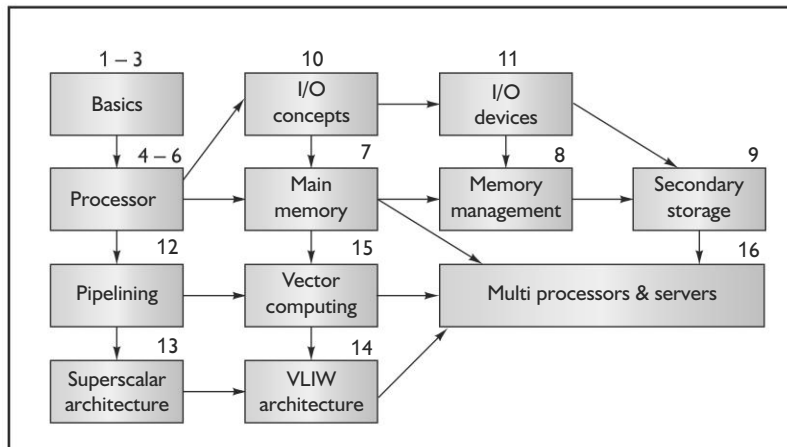
EXERCISES

1. A program has 20% branch instructions. Assuming that all the branches are successful, calculate the speedup, in a pipelined processor with five stages.
2. A program has 20% branch instructions. When it is run on a four-stage pipelined processor, it was observed that 60% of the branches were successful. Calculate the performance improvement in the pipelined processor. What is the maximum speedup possible if there are no branches?

CHAPTER 13



SUPERSCALAR ARCHITECTURE



Chapter Flow Diagram

OBJECTIVES

- « *Superscalar architecture*
- « *Parallelism*

KEYWORDS

Superscalar Architecture
Static Scheduling
Dynamic Scheduling
Out-of-Order Execution
Register Renaming
Speculative Execution
Reservation Station
Reorder Buffer

Instruction Pool
Micro-ops
Squashing
Retire Unit
Commitment Unit
Parallel Processing
Instruction Level Parallelism
Overlap

 13.1 Introduction

In chapter 12, we have discussed basic pipelining concepts which help the processor, to overlap the execution of multiple instructions. The amount of ILP achieved in a scalar processor is limited by pipeline hazards. In this chapter, we review the advanced techniques used in superscalar processors, to increase the amount of ILP. Basically, these techniques extend the basic pipelining concept either by hardware approach, or by software approach to exploit the ILP. Being at introductory level, this chapter outlines the basic concepts behind the hardware and software approaches. The software approach discussed in this chapter is also used in VLIW processors discussed in chapter 14.

 13.2 Concept of Superscalar Architecture

The superscalar architecture allows the execution of two or more instructions simultaneously, in different pipelines. As a result, more than one instruction gets completed in every clock cycle. Thus, the throughput of a superscalar processor is greater than that of a pipelined scalar processor, by twice or more. A superscalar processor may use RISC or CISC architecture. PowerPC is a RISC processor whereas Pentium 4 is a CISC processor. In some superscalar processors, the sequencing of instruction execution is static (during compilation), whereas in others it is dynamic (at run time).

Processors such as Intel Pentium 4 use dynamic, hardware based, approach to detect and exploit the ILP dynamically (at program run time). Processors such as Intel Itanium depend on compiler technology to detect parallelism, statically (at compilation time). However, there are some processors which use a combination of hardware and software approaches.

13.2.1 Requirements of Superscalar Processors

The requirements for a successful superscalar execution are as follows:

1. The instruction fetching unit should fetch more than one instruction at a time.
2. The instruction decoding logic should check whether the instructions are independent and hence executable, simultaneously.
3. There should be sufficient number of execution units so that several instructions can be processed, simultaneously. The execution units may be preferably pipelined.
4. The cycle time for each pipeline stage should match the cycle times for the fetching and decoding logic, so that pipelines work at optimum efficiency.

13.2.1.1 Superscalar Pipelines

It may appear that each stage in the pipeline takes the same amount of time. But, in practice, some stages take more time than others. Assume that the Fetch, Decode, and Save stages, each take 25ns to complete, but the Execute stage takes 50ns to complete, in a 4-stage pipeline. Then, it takes 200ns to complete an instruction using pipelining. The reason is that the other stages are unable to receive the next instruction until the Execute unit is ready to receive the next instruction. In a pipeline, the time spent in each stage is same for all stages and is determined by the slowest stage. Except the Execute stage, the other stages (Fetch, Decode, and Store) take only 25ns to complete, but they wait for 25ns, before moving to the next instruction. In other words, the instruction cycle consists of four stages each taking 50 ns. It takes 200ns to complete the first instruction, and then for every 50ns, one instruction is completed since the clock cycle is 50 ns. Fig.13.1 shows the stages in a processor with a single pipeline. Fig. 13.2 shows the basic concept of superscalar pipeline. This pipeline can fetch and execute multiple instructions in each clock cycle. The instruction issue buffer stores and retains the instructions during the period, when operands are being generated by previous instructions. On the other hand, it enables issuing those instructions, which are ready for issue, *out-of-order*. It issues multiple instructions in each clock cycle to the multiple functional units which operate in parallel. For achieving maximum possible efficiency, the superscalar pipeline is supported by several techniques such as branch prediction, speculative execution and out-of-order-execution etc. Fig. 13.3 illustrates the addition of one more pipeline that improves the productivity.

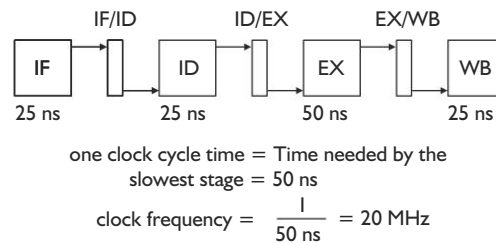


Fig. 13.1 Pipeline stages and clock cycle

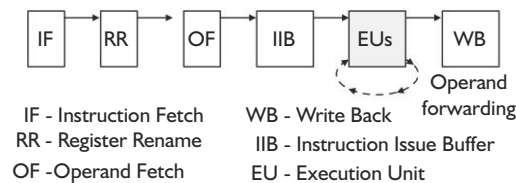


Fig. 13.2 Basic concept of superscalar pipeline

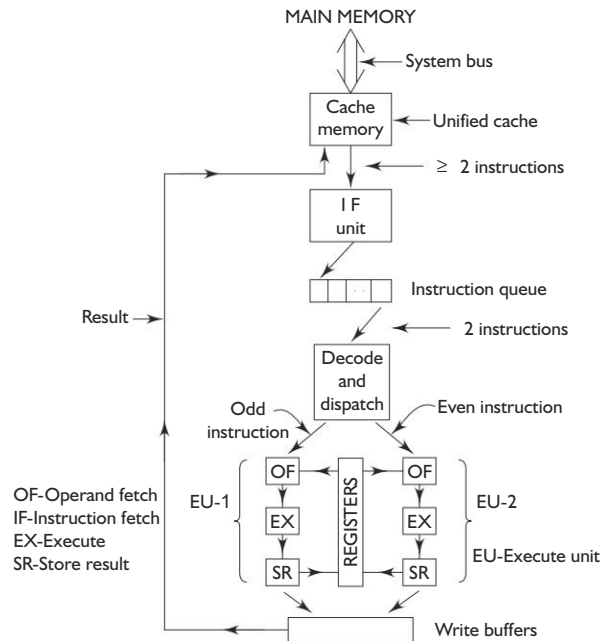
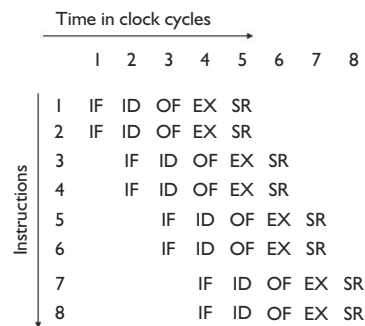


Fig. 13.3 A superscalar processor with two identical pipelines

The two execution units work independently. The odd instructions enter the execution unit 1, and the even instructions enter the execution unit 2. It takes 5 clock cycles to complete the first instruction and then for every clock cycle, two instructions are completed. Figure 13.4 shows the timing diagram for execution of instructions, in the program order, assuming no hazards are present. The instruction cycle has five stages. Fig. 13.5 shows the organization of a simple two-way superscalar processor.



Assumption: Alternate instructions are of different types. For example, instruction 1 may be ADD and instruction 2 may be LOAD

Fig. 13.4 Instruction execution in a 2-way superscalar processor when no hazards are present

It has one integer unit and one floating point unit both of which are pipelined. The integer instruction cycle has five stages, whereas the floating point instruction cycle has 7 stages. The floating point instruction has three execution phases: ex1, ex2 and ex3. If the object code contains integer and floating point instructions alternatively, two instructions can be completed in every clock from 7th clock onwards as shown in Fig.13.6. We can call this processor a 2-way superscalar processor, since two instructions can be executed in one clock cycle. In practice, it is difficult to achieve execution of two instructions in each clock cycle, because of dependencies and branch instructions.

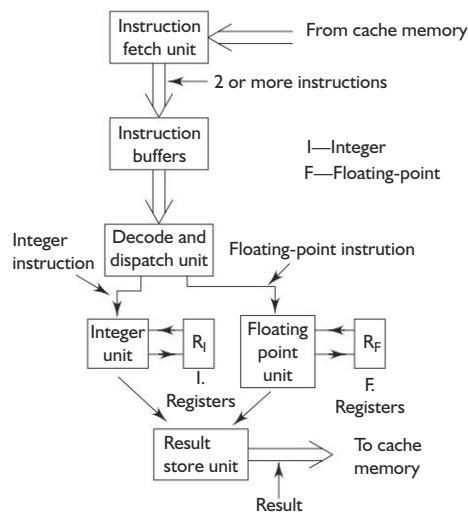


Fig. 13.5 Organization of a simple two-way superscalar processor

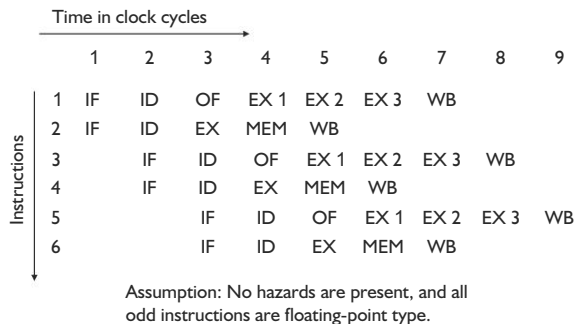


Fig. 13.6 Instruction execution in a 2-way superscalar processor

Table 13.1 summarizes various techniques followed in superscalar processors. Some of these are used in basic pipelining also, and have been discussed in Chapter 12. Intel Pentium is basic superscalar processor that follows only some limited techniques, whereas

Intel Pentium-pro is an advanced superscalar processor that follows several techniques. Though the presence of multiple functional units allows simultaneous execution of multiple instructions, often the functional units remain idle, without any instruction, due to dependencies and branch instruction execution.

TABLE 13.1 Superscalar techniques

Sl. no.	Technique	Basic principle	Objective
1	Operand forwarding	Required operand is directly made available at the required place (e.g. ALU), as soon as it is produced in a pipeline stage, even before it is stored in the destination	Reducing pipeline stall due to data hazard
2	Delayed branching	Placing an instruction unrelated to a branch instruction, immediately after the branch instruction (in the delay slot), by reordering the instructions in the object code	Reducing pipeline stall due to control hazard
3	Dynamic scheduling	Executing instructions out-of-order wherever required (and possible), without affecting program logic, instead of executing purely in-order	Reducing data hazard stall due to true data dependencies
4	Register renaming	Using multiple physical registers in the processor, against one logical (architectural) register, temporarily without affecting the final values in the logical register	Reducing (a) data hazard stall and (b) stall due to anti-dependency and output dependency
5	Branch prediction	Assuming either 'taken' branch or 'untaken' branch in advance, and controlling pipeline actions accordingly, on decoding a branch instruction, without waiting for the 'actual' fate of the 'branch'. Finally, corrective steps taken as applicable	Reducing pipeline stall due to control hazard
6	Multiple instruction issue	Issuing multiple instructions simultaneously to different pipelines either in-order, or out-of-order depending on processor capability	Reducing CPI by minimizing idle time of execution units
7	Speculative execution	Executing instructions in advance, both out-of-order and 'speculative', without knowing whether they need to be executed or not as per program logic. Finally, corrective steps taken as applicable	Reducing pipeline stalling due to data hazard and control hazard
8	Loop unrolling	Transforming a loop into instruction streams for various iterations and executing them simultaneously	Reducing pipeline stall due to control hazard
9	Software pipelining	Executing different iterations of a loop simultaneously, in a overlapped fashion	Exploiting ILP within a loop

13.3 Case Study 1 – Intel Pentium Processor

The Pentium is Intel's first superscalar processor with a CISC architecture. It has two integer execution units and one floating-point execution unit. A brief comparison of the Pentium features to its predecessor INTEL 80486 is given in Table 13.2.

The 80486 is a 32-bit processor with on-chip cache memory and on-chip Numeric Data Processor (NDP), a floating-point execution unit. Though it has two execution units (one integer and one floating-point), it is a scalar processor similar to Intel 80386 processor. However, it is more powerful than 80386 with several attractive features. The Pentium is a 32-bit processor for internal operations, but it has 64-bit external data bus. It can perform burst mode data transfer which is highly desirable, in view of internal cache memory.

TABLE 13.2 Performance Features: 80486 vs Pentium

Feature	486 implimentation	Pentium implimentation	Advantage in Pentium
Integer processor	Pipelined; scalar	Two pipelines: U and V. Superscalar.	Two instructions are simultaneously processed
On-chip floating-point unit	Non-pipelined; only adder in FPU	Pipelined FPU; Apart from adder, hardware multiplier and divider also in FPU	Floating-point multiplication and division are faster
On-chip Cache Memory	8 KB; unified	Separate 'code cache', and 'Data cache', each 8 KB	Resource conflict (cache memory) is reduced
Write output buffer	One set	One set each for U and V pipes	Resource conflict (memory bus) is reduced
Cache write policy	Write through	Write back	Number of memory writes are reduced improving resource (main memory) availability

Though Pentium's superscalar architecture offers performance improvement, dependency between two consecutive instructions affects the performance, since the pentium does not follow full fledged dynamic pipelining.

13.3.1 Superscalar Architecture in Pentium Processor

The Figure 13.7 shows the integer pipeline stages of a Pentium processor. The Pentium supports a two-way superscalar architecture with two integer pipelines, and one floating point pipeline. Though there are three execution units, the Pentium processor can not execute three instructions simultaneously. It can execute either two integer instructions together in one clock cycle, or one integer and one floating point instruction together in one clock cycle. Figure 13.8 illustrates the overall organization of the Pentium, and Fig. 13.9 depicts the implementation of the superscalar architecture in Pentium. The stages PF and D1 are common to both U and V pipelines. The D1 uses various techniques to decide if the given

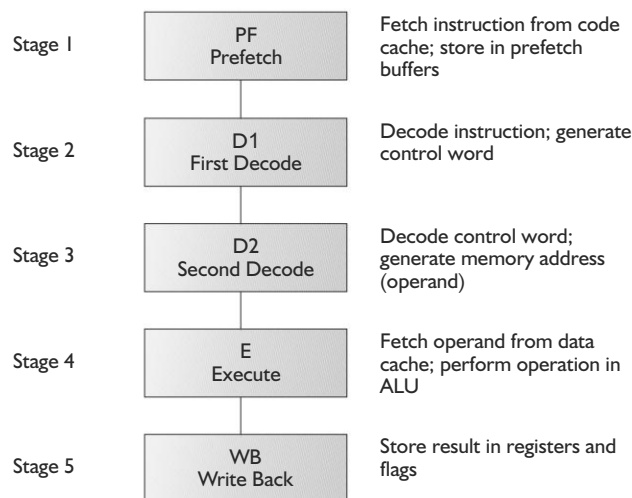


Fig. 13.7 Pentium integer pipelines stages

two consecutive instructions can be executed, simultaneously (in parallel), considering the type of instructions and dependencies between them. The U and V pipelines are identical except for the inclusion of some special hardware in U pipe (barrel shifter).

If both the instructions are of 'simple' types, they are issued to U and V pipes, one each. If one of these is a 'complex' instruction, it is sent to U pipe, since the V pipe is not capable of executing it. Also, a floating-point instruction can be executed only in U pipe, and during that time, the V pipe can not be given any instruction for execution. Similarly, when U pipe executes a complex instruction, V pipe's resources also may be used by U pipe. It can be easily observed that two complex instructions can not be executed simultaneously.

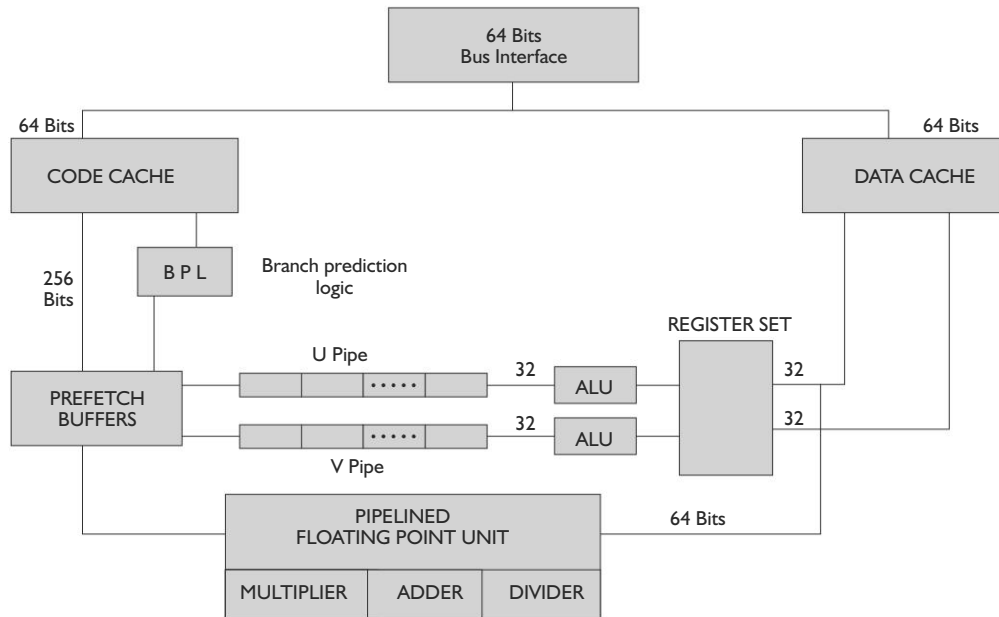


Fig. 13.8 Simplified organization of Pentium

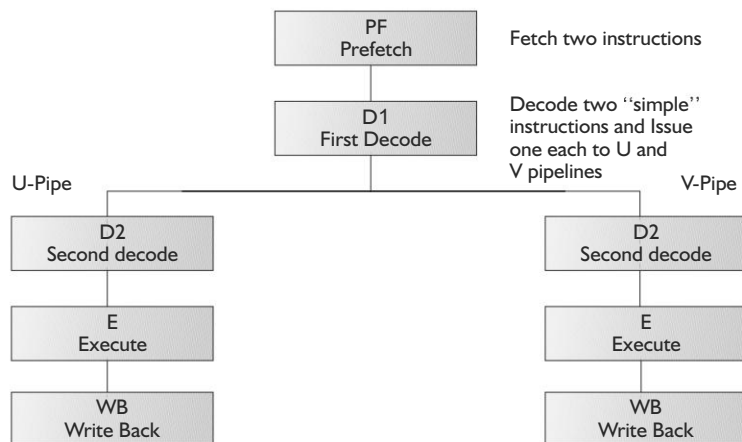


Fig. 13.9 Superscalar execution in Pentium

The floating-point pipeline has eight stages, as shown in Fig. 13.10. The first five stages (PF, D1, D2, E and X1) are nothing but the five stages of U pipe.

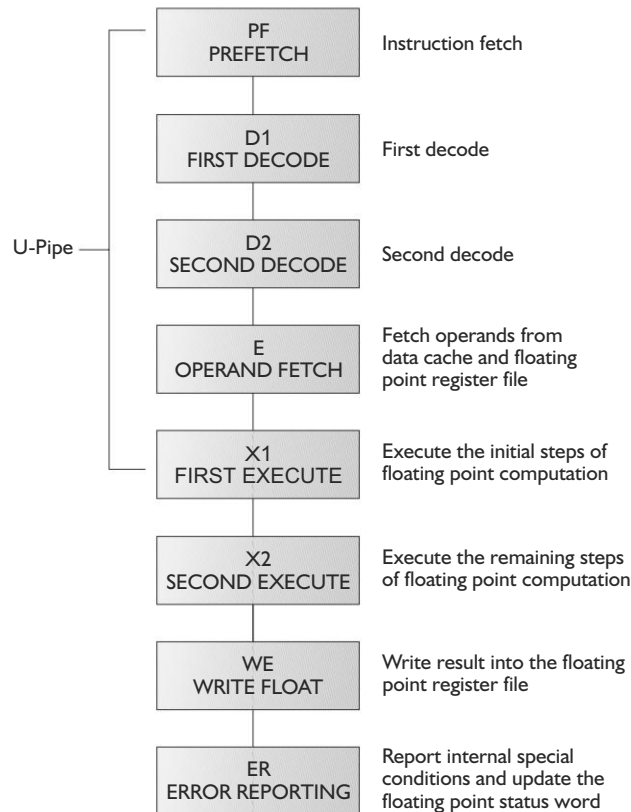


Fig. 13.10 Pentium floating-point pipeline

13.3.2 Pentium and Dynamic Branch Prediction

The Pentium uses prediction based on past history by using a Branch Target Buffer (BTB), an associate memory, in the processor. The BTB has 256 locations. It is used to store the history of branches, by entering the following information (Fig. 13.11), for every completed branch instruction:

1. Branch instruction address
2. Branch destination address
3. Branch history (yes/no)

Whenever a branch instruction is decoded, the Pentium searches the BTB, to check whether there is an entry, corresponding to the current branch instruction address and target address. If there is an entry in the BTB (i.e. HIT) with these two addresses, the processor is aware of the previous case. The HISTORY bit informs, if the branching occurred or

not in the last (previous) execution of this instruction, i.e. during previous iteration of a loop. The processor assumes similar situation now and acts accordingly. In case, the branch has to be taken, the microprocessor fetches instruction from the target address. This may not be a correct prediction for the current instruction, and 'the real fate' will be known only in the last step of the conditional branch instruction. If the prediction is wrong, the actions done based on it are ignored. In such cases, the processor modifies the history bit in the BTB.

Branch instruction address	Branch address	History Y/N
-------------------------------	-------------------	----------------

Fig. 13.11 Branch target buffer word format (Pentium Processor)



13.4 Out-of-Order Execution

Superscalar processors can complete the processing of instructions in the non-program order. Before the completion of earlier instructions, the later ones can be executed. This flexibility improves the performance. However, the results of the execution have to be rearranged in the correct order, to ensure that the program logic is not altered. This is usually carried out by the retirement unit, also known as commit unit.

13.4.1 Dynamic Scheduling

Instead of delaying all subsequent instructions in a pipeline, when one instruction is held up in pipeline due to a hazard, a superscalar processor can check if any subsequent instruction can be executed without hazard.

Consider the following instruction sequence:

ADD R2, R3

SUB R4, R2

MUL R5, R6

The SUB instruction depends on ADD instruction, since the second operand of SUB instruction is the result of ADD instruction. Hence, in a simple scalar processor, the pipeline has to be stalled while executing the SUB instruction. Because of this, the MUL instruction is also held up in the pipeline, even though it has no dependency issue with any earlier instruction. In a superscalar processor with dynamic pipelining, the MUL instruction can be executed, out-of-order, in another pipeline/execution unit. In dynamic scheduling, instructions are issued in-order but executed out-of-order. To enable out-of-order execution, the instruction decode (ID) stage is divided into two stages:

1. Issue: Decoding instructions, and checking for structural hazards
2. Operand fetch: Waiting till data hazard condition disappears

Figure 13.12 (a) shows basic concept of dynamic scheduled processor, and Fig.13.12(b) shows the basic concept of out-of-order execution. Fig.13.13 (a) shows the organization of a basic superscalar processor and Fig.13.13 (b) shows the block diagram of advanced superscalar processor.

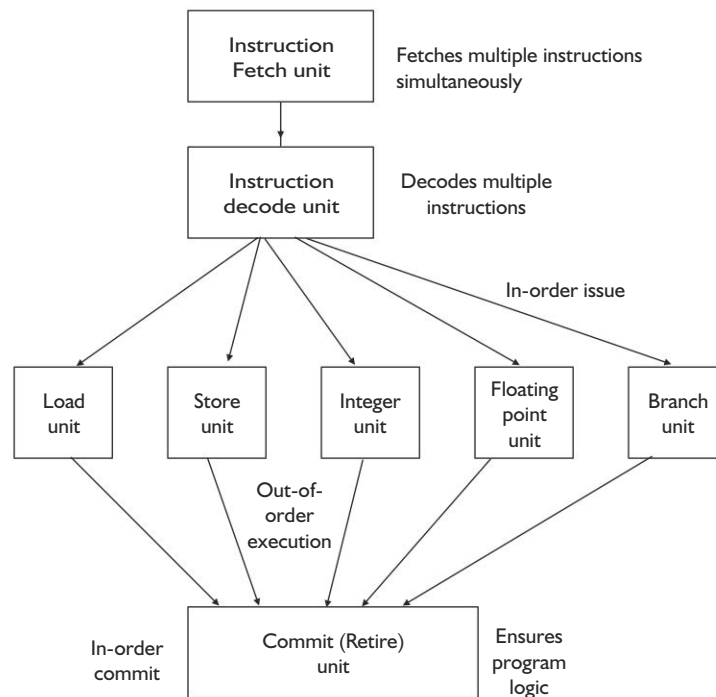


Fig. 13.12(a) Basic concept of dynamic scheduled processor

13.5 Register Renaming

Register renaming is a technique that eliminates the conflict between the different execution units, trying to use the same registers as required by the instructions. Instead of one set of architectural registers, multiple sets of physical registers are used in the processor. For each program addressable logical register, there are multiple physical registers. This enables the different execution units to work, simultaneously, without the pipeline stalls due to resource conflict. The renaming is a temporary requirement. Once the execution is complete, the result is put in the originally intended register, and the renaming is removed.

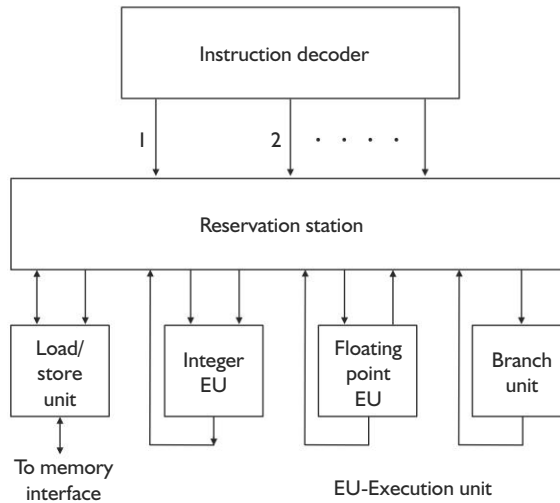


Fig. 13.12(b) Basic concept of out-order execution

13.5.1 Resolving Antidependency and Output dependency by Register Renaming

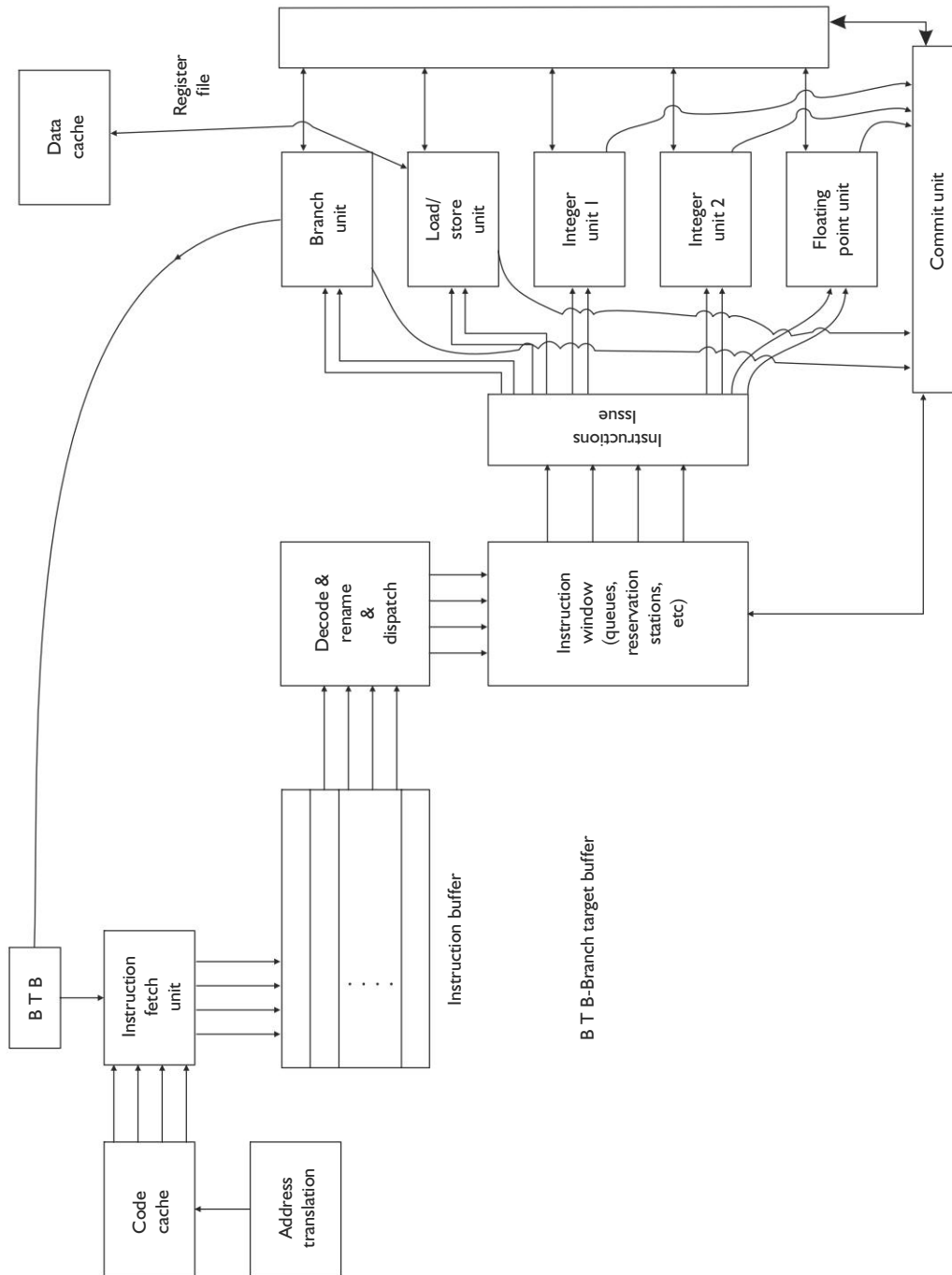
If two instructions are writing into the same location, an output dependency exists between them. If the later instruction writes before the earlier one, it affects the program logic. An antidependency exists if the earlier instruction uses a location as an operand, while a later instruction writes into that location. Consider following example:

- 1 MOVE R4, R9
- 2 LOAD R8, 0(R4)
- 3 ADD R4, R2, 3

An output dependency exists between the first and third instruction since both write in R4. An anti-dependency exists between the second and third instruction since the third instruction writes into R4 whereas the second instruction reads from R4, for memory address calculation. These two types are not true data dependencies since there is no data transfer between the instructions. The problem caused by these two dependencies can be resolved if we use one temporary register as shown below:

- 1 MOVE R4, R9
- 2 LOAD R8, 0(R4)
- 3 ADD R14, R2, 3

Use of R14 in the third instruction removes both anti dependency and output dependency. It is easily noticed that the program logic is not disturbed. This technique is known as register renaming. Finally, the content of R14 can be restored in R4 so that the program results are not affected.



B T B-Branch target buffer

Fig. 13.13(a) Block diagram of a basic superscalar processor

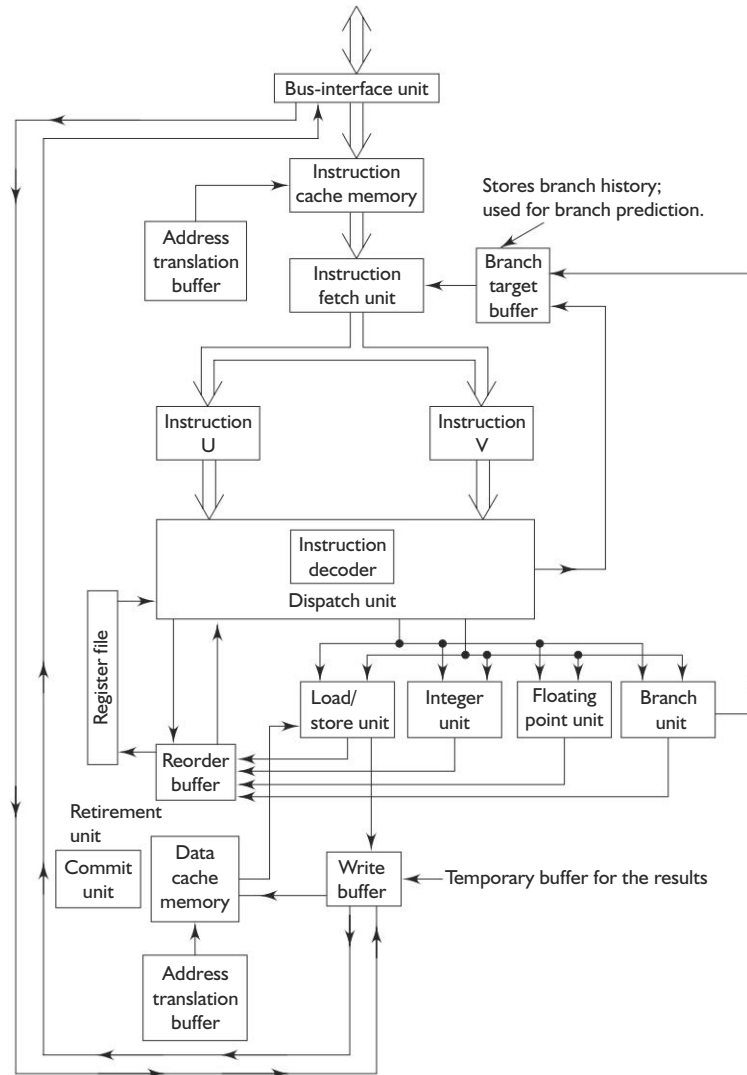


Fig. 13.13(b) Block diagram of an advanced superscalar processor



13.6 Speculative Execution and Branch Prediction

The superscalar processors have the ability to execute multiple instructions, simultaneously. In some cases, the results of the execution may not be used, because alteration in the program flow would imply that the instruction should not be fetched at all. In particular, this

occurs in the vicinity of branches, where a condition is tested and the program path is altered, depending on the branch results.

Branches are very common in the code and create a serious hindrance for pipelining. When we have a conditional test instruction (i.e. an “IF.. THEN” instruction), it is not known which should be the next instruction, until the condition test has been completed. A simple processor will stall the pipeline until the result is known, affecting the performance. Advanced processors speculatively execute the next instruction, with the assumption that it will use the result if the branch follows, the assumed path. In case, the assumption is wrong, the result based on speculative execution will be dropped. This action will be done during commit stage. Certain advanced superscalar processors combine this with branch prediction, where the processor predicts the accurate path based on the past history. This is relevant mostly for loops.

Let us consider the following source code:

- IF $I = J$ THEN
 $M = M + 1$
- ELSE
 $M = M - 1$

The “IF...THEN” is a branch. Until it is completely executed, it is not known, whether the next instruction will be an addition, or subtraction. In such cases, the processor uses the branch prediction to start only one of them, based on the *previous* information.

Branch prediction improves the handling of branches by making use of a small cache memory called the Branch Target Buffer (BTB) as in Pentium processor. Whenever the processor executes a branch, it stores information about the *branch result*. Again, when the processor encounters the same branch, it makes a *guess* about present path to be followed by the branch, based on earlier result. This enhances the pipeline performance.

Instead of mere branch prediction, a superscalar processor can speculatively execute all, the instructions, addition and subtraction, at the same time, and discard the useless.

13.7 Loop Unrolling

A basic block is an object code portion with no branching-out or entry-in, except at the end and at the beginning respectively. In a typical program, branch instructions are present for every three to six instructions on an average. Hence, the average block size is small and the amount of ILP in a block is also limited. Loop level parallelism (LLP) uses parallelism among iterations of a loop.

Loop Unrolling and Pipeline scheduling

Consider the following loop that adds a scalar constant to a 100 element array:

FOR ($i=100$; $i>0$; $i=i-1$)

$X[i] = x[i] + 5$

Assume following code is generated for this loop. R4 has the constant 5; R2 contains the array element. Initially R1 gives the 100th element's address. R3 contains the address of the first element. Each element is stored in 4 bytes.

```

LOOP: LOAD      R2, 0(R1)
      ADD       R2, R4
      STORE     R2, 0(R1)
      DEC       R1, - 4
      BNE      R1, R3, LOOP

```

Within each loop iteration, there is not much parallelism. But each iteration is independent. Hence, we can overlap any iteration of the loop with any other iteration. Thus, loop level parallelism can be converted into instruction level parallelism by unrolling the loop. In unrolling the loop, the loop body is replicated many times, as shown below for five iterations.

```

LOAD  R2, 0(R1)
ADD   R2, R4
STORE R2, 0(R1)
LOAD  R2, -4(R1)
ADD   R2, R4
STORE R2, -4(R1)
LOAD  R2, -8(R1)
ADD   R2, R4
STORE R2, -8(R1)
LOAD  R2, -12(R1)
ADD   R2, R4
STORE R2, -12(R1)
LOAD  R2, -16(R1)
ADD   R2, R4
STORE R2, -16(R1)

```

Loop unrolling allows instructions from different iterations to be scheduled simultaneously. Also, different registers can be used for different iterations. Though the memory requirement increases, execution time is reduced by a large extent, in a superscalar processor, due to parallelism. Loop level parallelism is the concept behind vector instructions. In

vector processors, the data items are operated in parallel. Chapter 15 discusses arithmetic pipelines used in vector processors.

Consider the execution of following loop:

```
Loop: LD      F2,0(R1)
      SUBD    F8,F2,F4
      MULTD   F10,F8,F0
      SUBI    R1,R1,#8
      BNEZ    R1,Loop
```

Integer registers are named as R0, R1, R2 etc. whereas floating point registers are named as F0, F2, F4 etc. Assume that the compiler unrolls the loop as follows:

```
Loop: LD      F2,0(R1)
      SUBD    F8,F2,F4
      MULTD   F10,F8,F0
      LD      F12,-8(R1)
      SUBD    F14,F12,F4
      MULTD   F16,F14,F0
      SUBI    R1,R1,#16
      BNEZ    R1,Loop
```

The ILP in the unrolled code can be easily exploited by the superscalar processor.



13.8 Dynamic Scheduling and Static Scheduling

The performance improvement of a superscalar processor depends on the execution of multiple instructions in parallel. The extent of instruction-level parallelism in a basic block of object code is usually small. If there is a way to look across the basic block boundaries, massive enhancement in performance can be achieved. For example, if the do-loop branches are resolved early, it exposes the parallelism between many iterations. There are two ways of extracting parallelism from a program: static extraction (compilation time), and dynamic extraction (run-time).

Statically-scheduled architecture exploits instruction-level parallelism by exposing the processor's parallel structure in the object code. Compilers arrange the parallelism across basic blocks by techniques such as software pipelining or trace scheduling.

Dynamically-scheduled architecture depends mostly on speculative execution of the processor. The side effects of speculative execution can be overcome by the processor by using the buffers and following a careful action plan at the final stage: commit or discard. The processor has extra hardware logic for performing the following actions:

1. Looking far ahead in the instruction stream and identifying independent operations.
2. Scheduling the independent operations out of order.

Superscalar architecture generally follows dynamic scheduling whereas VLIW architecture uses static scheduling. Certain superscalar architectures also follow static scheduling and the processor hardware is unaware of it. The static scheduling is discussed in chapter 14 (VLIW architecture).

13.8.1 Dynamic Scheduling

The dynamically scheduled superscalar processor isolates the instruction fetch/decode, from instruction issue/execute, so that each operates at its own pace. The processor handles the object code motion across basic block boundaries by multiple techniques:

- looking ahead in the instruction stream
- buffering the internal state
- speculative execution of instructions
- Out-Of-Order (OOO) execution of instructions

The processor issues independent instructions from a *window* of dynamic instructions. To maintain the program logic, instruction scheduling is performed by the processor from this window. Out-of-order execution and branch prediction help the processor, in executing the instructions from multiple basic blocks, simultaneously. Buffering within the processor supports the speculative execution of some instructions before the execution of previously fetched branch instructions.

The disadvantage of dynamic scheduling are:

1. The hardware complexity is increased due to detection and scheduling of independent operations.
2. There is a limit on the parallelism since the hardware can analyze only a small window of dynamic instructions during each cycle.
3. The number of ‘useful’ instructions fetched per cycle are reduced due to the large number of branches during the execution.

13.8.2 Dynamic Scheduling Algorithms

Recognizing the presence of conflicts and resolving them is essential, so that the program logic is undisturbed. Interesting techniques were developed by Thornton (for Control Data CDC6600), and Tomasulo (for floating point in IBM 360/91) for resolving the dependencies. Both these CPUs were pipelined, and supported out-of-order execution. The CDC 6600 had a load-store architecture. As these systems are old, they are not viewed as

superscalar by many professionals. However, many of the olden techniques in these are implemented in the modern pipelined and superscalar processors.

13.8.2.1 Reorder Buffers and Register Renaming

The *true* data dependency is due to program logic, whereas the name dependency (register reference) is due to architectural limitations.

Two common techniques used for resolving dependencies are:

1. *Register renaming*: In the register renaming principle, the processor has a large set of registers. These are dynamically allocated to the architectural registers as and when required. There are several versions of an architecture register present at a time.
2. Using a reorder buffer.

Reorder Buffers

This technique is based on a queue system. It uses a physical register file that is of same size as the architectural register file. In addition, it has a set of registers arranged as a queue data structure, known as the reorder buffer.

When an instruction is issued, an entry for its results is assigned, at the *tail* of the reorder buffer. The logical order of instructions (as in the program) is preserved within this buffer. As the instruction execution proceeds, the assigned entry is updated, indicating the status of the instruction. In practice, the reorder buffer entries need to maintain some information about the instruction results i.e. the instruction, destination and validity of the result. When an entry reaches the *head* of the reorder buffer, it is removed if it has been entered with its result. The results are transferred from the temporary registers to the architectural (permanent) registers, before removing the instruction from the reorder buffer. If the result is not yet available, the entry waits. On removing an instruction from the reorder buffer, temporary registers and other hardware resources assigned to that instruction are released. It is possible that the reorder buffer entry at the head of the queue is still waiting, but some subsequent entry is ready. The later entry stays in the reorder buffer until the head instruction is completed. But, its result can be used if required by some other instruction. This process is called forwarding. An instruction can retire only after reaching the head of the queue. Hence, all previously issued instructions should have already retired. This ensures in-order retirement of instructions.

Reorder buffer method has introduced an extra step to the pipeline: transferring the result from reorder buffer to architectural registers. Also, a reorder buffer is an additional *place* where the execution units must check for operands, in addition to the actual registers.

Precise Architectural States

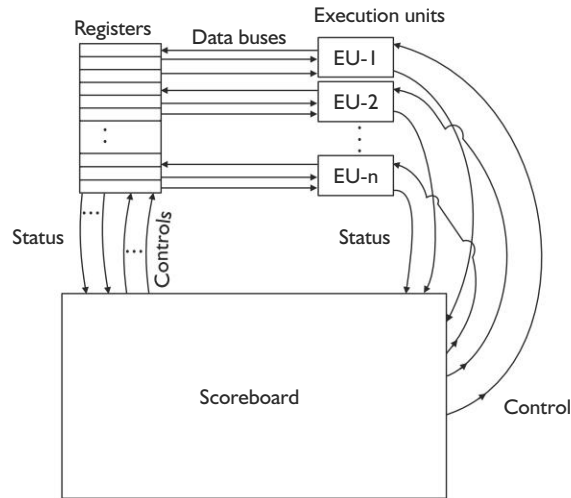
The maintenance of a precise architectural state is automatically achieved in a reorder buffering. Consider that an instruction $n + 1$ has been completed before the instruction n , and subsequently the instruction n results in an error (exception). The reorder buffers can solve this by keeping the instruction results provisional until the earlier ones are known, provided the instructions are actually issued in the program order. Thus, precise architectural state is guaranteed. Assuming that the instruction issue is in the program order, the order of the entries in the reorder buffer reflects the program order. In case of an exception occurring during an instruction execution, all later instructions that have been already executed partly must be omitted, to satisfy *precise exception* handling.

13.9 Thornton Technique and Scoreboard

Introduced in the CDC 6600, scoreboarding is a centralized method for dynamically scheduling a pipeline, to execute instructions out-of-order. The scoreboard is a central hardware logic where information about the currently active instructions is maintained. The scoreboard monitors the status of each instruction waiting to be dispatched. After it determines that all the source operands, and the required functional unit are available, it dispatches the instruction for execution. The scoreboard is totally responsible for instruction issue and execution including hazard detection. The objective of the scoreboard is executing an instruction as soon as possible. It decides when an instruction begins and ends execution. It keeps track of the progress of instructions, and the status of the hardware resources in the CPU (all functional units and registers). Figure 13.14 (a) shows the basic organization of a scoreboard based system.

In a scoreboard, the data dependencies of all active instructions in the pipeline are entered. An instruction is issued only when the scoreboard ensures that there are no conflicts with previously issued and incomplete instructions. If an instruction is stalled, the scoreboard monitors the flow of currently executing instructions until all dependencies have been resolved, before the stalled instruction is issued. In a scoreboard based system, as shown in Fig. 13.14 (b), instructions go through four major stages: Issue, Read Operands, Execution and write result.

1. *Issue stage:* The scoreboard tests for a free functional unit, and for the presence of WAW hazards. If it finds a WAW hazard, or no functional unit is free for the instruction, the instruction stalls. The scoreboard also checks which are source registers, and which is destination register for this instruction. This information is stored for use in the following stages.



CDC 6600 has 10 functional units and 24 architectural registers. Upto 10 instructions can be issued to the processor

Fig. 13.14(a) Basic organization of a superscalar processor with a scoreboard

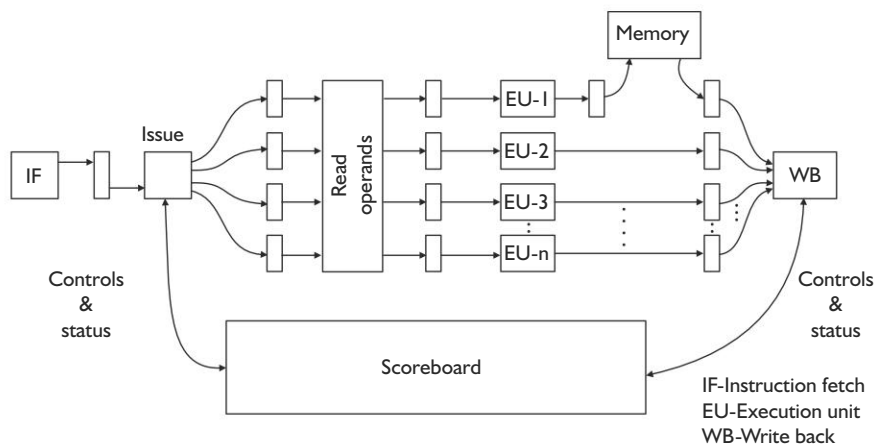


Fig. 13.14(b) Instruction stages in a superscalar processor with scoreboard

- (a) To avoid WAW hazard, the instruction is stalled, until all earlier instructions, (currently in the pipeline), going to write to the same destination register, are completed.
 - (b) The instruction is stalled when required functional units are currently busy.
- The instruction issue takes place when both the conditions are satisfied.
2. *Read operands stage:* This step resolves RAW hazards. After an instruction has been issued and assigned to the required functional unit, the instruction is made to wait until all operands become available. The scoreboard checks for the availability of the source operands. If the source operands are available, the scoreboard directs the functional unit to fetch the operands from the registers and commence execution. An operand is considered available, if no currently active instruction will generate it, or if it is currently being stored in the register file. This step resolves RAW hazards because registers which will be written by earlier instructions, in pipeline, are not considered available, until they are actually written.
 3. *Execution stage:* When both operands have been fetched, the functional unit starts its execution. After the result is ready, the functional unit reports to the scoreboard that it has completed execution.
 4. *Write result stage:* In this stage, the result has to be stored in its destination register. However, this operation is delayed to resolve WAR hazards. Once the scoreboard receives the status that a functional unit has finished execution, it tests for any WAR hazards, i.e if any earlier instruction which is still in the 'read operands' stage has the current instruction's destination register, as one of its source operands. If a WAR hazard is present, the scoreboard directs the functional unit to stall until the hazard disappears. During this period, the functional unit is unavailable to other instructions.

To monitor the execution of the instructions, the scoreboard maintains three status tables:

1. *Instruction Status:* Indicates, for each instruction being executed, which of the four stages it is in. Each instruction, that has been issued or pending issue, is entered in this table. Fig. 13.15 gives the format of instruction status.
2. *Functional Unit Status:* Indicates the state of each functional unit as shown in Fig. 13.16. Table 13.3 gives the definition of the nine fields. There is one entry for each functional unit in this table. Once an instruction is issued, the status of its operands is maintained in this table.
3. *Register Status:* Indicates, for each register, which functional unit will store results into it. Fig. 13.17 shows the format of register status. The number of entries is equal to the number of registers.

Instruction	Issue	Read operands	Execution Complete	Write result

Fig. 13.15 Format of instruction status in scoreboard

Functional unit	Busy	OP	Fi	Fj	Fk	Qj	Qk	Rj	Rk

Fig. 13.16 Format of functional unit status in scoreboard**TABLE 13.3** Meaning of bits in functional unit status (Scoreboard)

Status	Meaning
Busy	FU is busy or not (Yes/No)
OP	Operation to be performed in the FU e.g. a Multiplier can perform multiplication or division; an adder can perform addition or subtraction
Fi	Destination register number
Fj, Fk	Source registers
Qj	Functional unit producing a result for Fj
Qk	Functional unit producing a result for Fk
Rj	'Yes' indicates if Fj is ready and not read; 'no' indicates if Fj has been read
Rk	'Yes' indicates if Fk is ready and not read; 'no' indicates if Fk has been read

	R1	R2	R3	R4		R12
Functional unit		Adder 1		Multiplier 1		

Fig. 13.17 Register result status format (Scoreboard)

Note: Blank indicates there is no pending instruction that will use this register as destination.

Example 13.1 Scoreboard operation

Consider the execution of following sequence of instructions is to be executed on a scoreboard system. Assume that our superscalar processor has 2 integer units, 2 FP add units, 1 FP multiply unit, and 1 FP divide unit.

```
Loop: LD      F2,0(R1)
      SUBD    F8,F2,F4
      MULTD   F10,F8,F0
      SUBI    R1,R1,#8
      BNEZ    R1,Loop
```

Integer registers are named as R0, R1, R2 etc. whereas floating point registers are named as F0, F2, F4 etc.

Instruction set of MIPS64 has been used here. It is 64-bit version of the instruction set. Instructions have D on the start or end of the mnemonic. For example, MULTD is the 64-bit version of the multiply instruction; LD is the 64-bit version of the load instruction. Assume that on unrolling the loop, we get following code:

```
Loop: LD      F2,0(R1)
      SUBD    F8,F2,F4
      MULTD   F10,F8,F0
      LD      F12,-8(R1)
      SUBD    F14,F12,F4
      MULTD   F16,F14,F0
      SUBI    R1,R1,#16
      BNEZ    R1,Loop
```

A. Scoreboard status after issuing five instructions

The scoreboard's initial status just after issuing five instructions is shown in Fig.13.18 to Fig.13.20. In the initial status of the tables, it has been assumed that all the instructions had been included in the instruction status table. We issue instructions sequentially until we face a WAW hazard or we exhaust functional units. In this case, we have exhausted functional units. Note that we have issued five instructions simultaneously, in the same clock cycle.

B. Scoreboard status after first two instructions completion

Figures 13.21 to 13.23 show the status of the scoreboard after the first SUBD has stored its result in the register file.

Instruction	Issue	Read operands	Execution	Write result
LD F2,0(R1)	✓	-	-	-
SUBD F8,F2,F4	✓	-	-	-
MULTD F10,F8,F0	✓	-	-	-
LD F12,-8(R1)	✓	-	-	-
SUBD F14,F12,F4	✓	-	-	-
MULTD F16,F14,F0	-	-	-	-
SUBI R1,R1,#16	-	-	-	-
BNEZ R1,Loop	-	-	-	-

Fig. 13.18 Instruction status just after issuing five instructions

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int1	Yes	Load	F2	R1	-	-	-	No	-
Int2	Yes	Load	F12	R1	-	-	-	No	-
Add1	Yes	Sub	F8	F2	F4	Int1	-	No	Yes
Add2	Yes	Sub	F14	F12	F4	Int2	-	No	Yes
Mult1	Yes	Mul	F10	F8	F0	Add1	-	No	Yes
Div1	No	-	-	-	-	-	-	-	-

Fig. 13.19 Functional unit status just after issuing five instructions

F0	F2	F4	F6	F8	F10	F12	F14	F16
-	Int1	-	-	Add1	Mult1	Int2	Add2	-

Fig. 13.20 Register result status just after issuing five instructions

Instruction	Issue	Read operands	Execution	Write result
LD F2,0(R1)	✓	✓	✓	✓
SUBD F8,F2,F4	✓	✓	✓	✓
MULTD F10,F8,F0	✓	-	-	-
LD F12,-8(R1)	✓	✓	✓	✓
SUBD F14,F12,F4	✓	✓	✓	-
MULTD F16,F14,F0	-	-	-	-
SUBI R1,R1,#16	-	-	-	-
BNEZ R1,Loop	-	-	-	-

Fig. 13.21 Instruction status after first two instructions completion

As shown in Fig.13.22, Int1, Int2, Add1 and Div1 are free and the multiply unit is busy. But, the next pending instruction is a multiply. The second add unit has almost completed; it is one clock cycle behind the first add unit since they both need to access the F4 register from the register file. Also, now Rj and Rk both indicate 'Yes' for the Mul1 unit, which means the operands are ready and hence the Mul1 unit can read the operands in the next clock cycle.

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int1	No	-	-	-	-	-	-	-	-
Int2	No	-	-	-	-	-	-	-	-
Add1	No	-	-	-	-	-	-	-	-
Add2	Yes	Sub	F14	F12	F4	-	-	No	No
Mult1	Yes	Sub	F10	F8	F0	Add1	-	Yes	Yes
Div1	No	-	-	-	-	-	-	-	-

Fig. 13.22 Functional unit status after first two instructions completion

The scoreboard system executes two iterations of the loop in parallel, since there are no loop-carried dependences. However, it will be limited by available functional units. After the operands are available, the execution times for various operations can be assumed as follows: loads - 4 cycles, integer ALU operations - 3 cycles, floating point add - 4 cycles, and floating point multiply - 9 cycles. For these execution specifications, the scoreboard system will take 26 cycles complete the above instructions.

F0	F2	F4	F6	F8	F10	F12	F14	F16
-	-	-	-	-	Multi	-	Add2	-

Fig. 13.23 Register result status after first two instructions completion



13.10 Tomasulo Algorithm and Reservation Stations

The Tomasulo algorithm is a technique for dynamic scheduling developed for the IBM System/360 Model 91's floating point unit by Robert Tomasulo. It follows out-of-order execution of sequential instructions. This algorithm is based on register renaming technique. Where as scoreboarding resolves WAW and WAR hazards by stalling, register renaming in Tomasulo algorithm allows the continual issuing of instructions.

TABLE 13.4 Meaning of fields in reservation station (Tomasulo's algorithm)

Field	Meaning
OP	Operation to be performed on source operands S1 and S2
Qj	Reservation station that produces the source operand S1; if S1 is not needed or already available in Vj, Qj will be made 0.
Qk	Reservation station that produces the source operand S2; if the 2 is not needed or already available in Vk, Qk will be made 0.
Vj	Value of S1
Vk	Value of S2
A	For a load or store instruction, information relevant to memory address calculation. (Initially before the address calculation, A is same as the immediate field in the instruction; after the address calculation, A gives the effective address)
Busy	This reservation station and the corresponding functional unit are busy

Basic concepts of Tomasulo's Algorithm:

- Instructions are issued sequentially (though they are executed out-of-order); This ensures that effects of exceptions encountered by these instructions is same as they would in a non-pipelined processor.
- Functional units use reservation stations with multiple slots. Each slot holds information needed to execute a single instruction, including the operation and the operands. Tomasulo's algorithm implements register renaming through the use of reservation stations. Reservation stations are buffers which fetch and store instruction operands as soon as they're available. Each reservation station corresponds to one instruction. The functional unit begins processing when it is free, and when all source operands needed for an instruction are available.
- The general-purpose and reservation station registers hold either real or virtual values of operands. If the content of a source register is not known, during the instruction issue stage, a *virtual value* is initially used. The functional unit that will generate the real value is assigned as the virtual value. The virtual register values are converted to real values, as soon as the relevant functional unit generates its result. The Tomasulo algorithm uses a common data bus (CDB) on which, results are broadcast to all the reservation stations that may need it. This enables early commencement of instructions waiting for these results as operands. Fig.13.24 (a) and Fig.13.24 (b) show the format of reservation station and register status respectively. Table 13.4 gives the meaning of different fields of the reservation station.

Name	OP	Qj	Qk	Vj	Vk	A	Busy
Adder 1							
Adder 2							

Fig. 13.24(a) Format of reservation station (Tomasulo's algorithm)

clock	Field	R1	R2	R3	R4		R12
	Qi		Adder 1		Multiplier 1		

Fig. 13.24(b) Register status (Tomasulo's algorithm)

Q_i : A field in the register file indicating its status. It identifies the reservation station, that has the operation whose result will be stored in this register. If no active instruction has this register as destination, Q_i is blank indicating that the register contains a valid value.

Fig.13.25 shows basic organization of a system using Tomasulo's algorithm. Each instruction passes through three stages: Issue, execute and write result

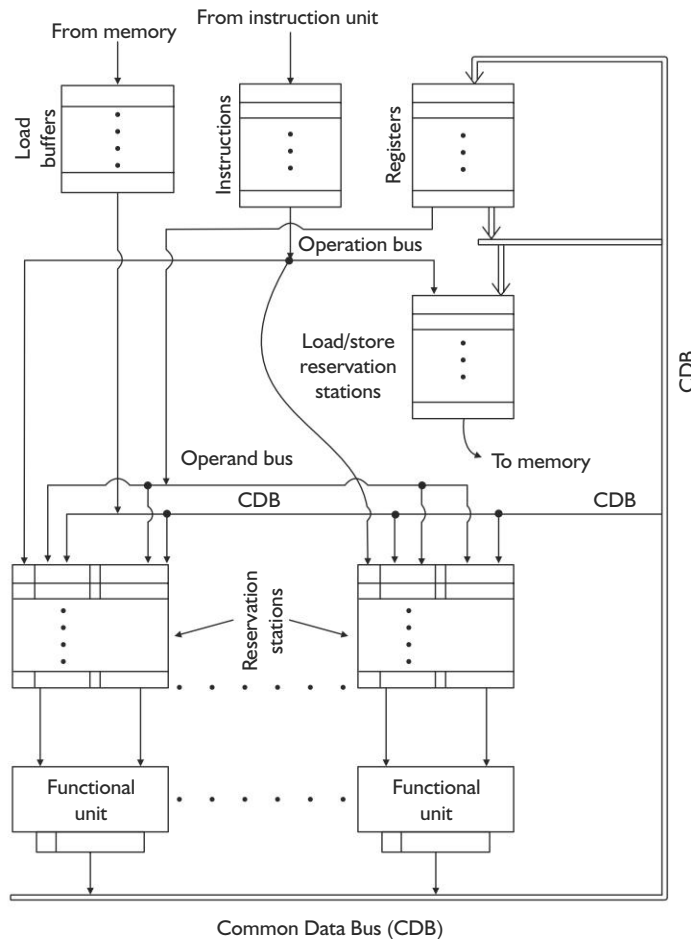


Fig. 13.25 Basic organization of a superscalar processor using tomasulo's algorithm

Stage 1: Issue stage

In the issue stage, an instruction is issued for execution if source operands and reservation station are ready; otherwise it is stalled. Registers are renamed in this step, eliminating WAR and WAW hazards. The algorithm used by the issue stage is as follows.

(a) Receive the next instruction from the head of the instruction queue (buffer). (b) If the operands are currently available in the architectural registers, check for the reservation station (b1) If there is a matching empty reservation station (i.e., functional unit is available), then issue the instruction. (b2) If there is not a matching empty reservation station (i.e., functional unit is not available), then stall the instruction until a station or buffer is free (c) If the operands are not available in the registers, the functional units which generate them are marked, to keep track of the functional units that will produce the operands.

Stage 2: Execute stage

In this stage, required functional operation is carried out. Instruction is delayed in this step until both operands are available; this eliminates RAW hazards. If any operand is not available then wait till the operand is available on the CDB.

When all operands are available, (a1) for the load or store instruction, compute the effective address when the base register is available, and place it in the load/store buffer; (a2) for the load instruction, execute as soon as the memory unit is available; (a3) for the store instruction, wait for the value (b). For ALU instruction, execute the instruction at the corresponding functional unit

Once execution is complete, the result is buffered at the reservation station.

Stage 3: Write result stage

In this stage, (a) For ALU operation, result is stored in registers; (b) For store operation, result is stored in memory.

For ALU operation, when the result is available, it is issued on the CDB, and from there, into the registers and any reservation stations waiting for this result. When the result is sent to the reservation stations, renaming occurs which eliminates antidependency and output dependency. For store instruction, result is stored in memory during this step.

WAW hazards are resolved since only the last instruction (in program order) actually stores in the destination registers. The other results are buffered in other reservation stations, and are eventually sent to any instructions waiting for those results. WAR hazards are resolved since reservation stations can get source operands from either the register file or other reservation stations (in other words, from another instruction).

Scoreboarding Vs Tomasulo's Algorithm

1. In Tomasulo's algorithm, the control logic is distributed among the reservation stations, whereas in scoreboarding, the scoreboard keeps track of everything.

2. In scoreboarding, the functional unit stalls during a WAR hazard; in Tomasulo's algorithm, the functional unit is free to execute another instruction. The reservation station sends the result to the register file and to any other reservation station which is waiting for that result.

Example 13.2 Tomasulo's Algorithm

Consider following instruction sequence.

```
LD      F8, 3(R2)
LD      F4, 4(R3)
MULTD   F2, F4, F8
SUBD    F10, F8, F4
DIVD    F12, F2, F8
ADDD    F8, F10, F4
```

Assume the number of execution cycles required by the functional units as follows: Integer: 1, FP Add: 2, FP Multiply: 10, FP Divide: 40. Assume following number of Reservation Stations: FP Add: 3, FP Multiply: 2, FP Load: 2, FP Store: 2

Integer registers are named as R0, R1, R2 etc. whereas floating point registers are named as F0, F2, F4 etc.

The progress of instructions during different clock cycles till cycle 6 are shown in Fig.13.26 to Fig.13.49. The reader is expected to continue the algorithm and verify the final results at the end of clock cycle 56, with the results shown in Fig.13.50 to 13.52.

	Issue	Execute	Write Result
LD F8, 3(R2)	I		
LD F4, 4(R3)			
MULTD F2, F4, F8			
SUBD F10, F8, F4			
DIVD F12, F2, F8			
ADDD F8, F10, F4			

Fig. 13.26 Instruction status (Cycle No. 1)

Name	Busy	Address
Load1	yes	(R2) + 3
Load2	no	
Store1	no	
Store2	no	

Fig. 13.27 Buffers (Cycle No.1)

Resv. Station	Busy	Op	Vj	Vk	Qj	Qk
Add1	no					
Add2	no					
Add3	no					
Mult1	no					
Mult2	no					

Fig. 13.28 Reservation station status (Cycle No. 1)

Field	F8
Qi	Load I

Fig. 13.29 Register result status (Cycle No. 1)

	Issue	Execute	Write Result
LD F8, 3 (R2)	1	2	
LD F4, 4(R3)	2		
MULTD F2, F4, F8			
SUBD F10, F8, F4			
DIVD F12, F2, F8			
ADDD F8, F10, F4			

Fig. 13.30 Instruction status (Cycle No. 2)

Name	Busy	Address
Load1	yes	$(R2) + 3$
Load2	yes	$(R3) + 4$
Store1	no	
Store2	no	

Fig. 13.31 Buffers (Cycle No. 2)

Resv. Station	Busy	Op	V _j	V _k	Q _j	Q _k
Add1	no					
Add2	no					
Add3	no					
Mult1	no					
Mult2	no					

Fig. 13.32 Reservation station status (Cycle No. 2)

Field	F4	F8
Q _i	Load2	Load1

Fig. 13.33 Register result status (Cycle No. 2)

	Issue	Execute	Write Result
LD F8, 3 (R2)	1	2	3
LD F4, 4(R3)	2	3	
MULTD F2, F4, F8	3		
SUBD F10, F8, F4			
DIVD F12, F2, F8			
ADDD F8, F10, F4			

Fig. 13.34 Instruction status (Cycle No. 3)

Name	Busy	Address
Load1	no	
Load2	yes	(R3) + 4
Store1	no	
Store2	no	

Fig. 13.35 Buffers (Cycle No. 3)

Resv. Station	Busy	Op	Vj	Vk	Qj	Qk
Add1	no					
Add2	no					
Add3	no					
Mult1	yes	MULTD		regs[F8]	Load2	0
Mult2	no					

Fig. 13.36 Reservation station status (Cycle No. 3)

Field	F2	F4
Qi	MultI	Load2

Fig. 13.37 Register result status (Cycle No. 3)

		Issue	Execute	Write Result
LD	F8, 3 (R2)	1	2	3
LD	F4, 4(R3)	2	3	4
MULTD	F2, F4, F8	3		
SUBD	F10, F8, F4	4		
DIVD	F12, F2, F8			
ADDD	F8, F10, F4			

Fig. 13.38 Instruction status (Cycle No. 4)

Name	Busy	Address
Load1	no	
Load2	no	
Store1	no	
Store2	no	

Fig. 13.39 Buffers (Cycle No. 4)

Resv. Station	Busy	Op	Vj	Vk	Qj	Qk
Add1	yes	SUBD	regs[F8]	regs[F4]	0	0
Add2	no					
Add3	no					
Mult1	yes	MULTD	regs[F4]	regs[F8]	0	0
Mult2	no					

Fig. 13.40 Reservation station status (Cycle No. 4)

Field	F2	F10
Qi	Mult1	Add1

Fig. 13.41 Register result status (Cycle No. 4)

	Issue	Execute	Write Result
LD F8, 3 (R2)	1	2	3
LD F4, 4(R3)	2	3	4
MULTD F2, F4, F8	3		
SUBD F10, F8, F4	4		
DIVD F12, F2, F8	5		
ADDD F8, F10, F4			

Fig. 13.42 Instruction status (Cycle No. 5)

Name	Busy	Address
Load1	no	
Load2	no	
Store1	no	
Store2	no	

Fig. 13.43 Buffers (Cycle No. 5)

Resv. Station	Busy	Op	V _j	V _k	Q _j	Q _k
Add1	yes	SUBD	regs[F8]	regs[F4]	0	0
Add2	no					
Add3	no					
Mult1	yes	MULTD	regs[F4]	regs[F8]	0	0
Mult2	yes	DIVD		regs[F8]	Mult1	0

Fig. 13.44 Reservation station status (Cycle No. 5)

Field	F2	F10	F12
Q _i	Mult1	Add1	Mult2

Fig. 13.45 Register result status (Cycle No. 5)

		Issue	Execute	Write Result
LD	F8, 3 (R2)	1	2	3
LD	F4, 4(R3)	2	3	4
MULTD	F2, F4, F8	3		
SUBD	F10, F8, F4	4	6	
DIVD	F12, F2, F8	5		
ADDD	F8, F10, F4	6		

Fig. 13.46 Instruction status (No. 6)

Name	Busy	Address
Load1	no	
Load2	no	
Store1	no	
Store2	no	

Fig. 13.47 Buffers (Cycle No. 6)

Resv. Station	Busy	Op	Vj	Vk	Qj	Qk
Add1	yes	SUBD	regs[F8]	regs[F4]	0	0
Add2	yes	ADDD		regs[F4]	Add1	0
Add3	no					
Mult1	yes	MULTD	regs[F4]	regs[F8]	0	0
Mult2	yes	DIVD		regs[F8]	Mult1	0

Fig. 13.48 Reservation station status (Cycle No. 6)

Field	F2	F8	F10	F12
Qi	Mult1	Add2	Add1	Mult2

Fig. 13.49 Register result status (Cycle No. 6)

	Issue	Execute	Write Result
LD F8, 3(R2)	1	2	3
LD F4, 4(R3)	2	3	4
MULTD F2, F4, F8	3	14	15
SUBD F10, F8, F4	4	6	7
DIVD F12, F2, F8	5	55	56
ADDD F8, F10, F4	6	9	10

Fig. 13.50 Instruction status (Cycle No. 56)

Name	Busy	Address
Load1	no	
Load2	no	
Store1	no	
Store2	no	

Fig. 13.51 Buffers (Cycle No. 56)

Resv. Station	Busy	Op	V _j	V _k	Q _j	Q _k
Add1	no					
Add2	no					
Add3	no					
Mult1	no					
Mult2	no					

Fig. 13.52 Reservation station status (Cycle No. 56)

No Entries

Fig. 13.53 Register result status (Cycle No. 56)

13.11 CASE STUDY 2 – Intel Pentium-pro Processor

The Pentium-pro is a CISC processor in view of its instruction set, but internally it follows the RISC techniques in the post-decode stages.

As shown in Fig. 13.54, the Pentium-pro is organized into three independent engines coupled with an instruction pool:

1. Fetch/Decode unit
2. Dispatch/Execute unit
3. Retire unit

The processor uses the out-of-order execution strategy so that it reduces idle time due to dependencies between the consecutive instructions. The instruction decoder splits each instruction into multiple simple operations called the *micro-ops*. All micro-ops are of uniform length (118 bits). The dispatch and parallel execution of the micro-ops of multiple instructions are carried out, simultaneously. To avoid clash between the registers specified in the instructions, register renaming technique is used. This involves reassignment of the logical address references to the physical registers. For this purpose, the Pentium-pro has 40 extra GPRs, in addition to the eight integer and eight floating-point registers (as per the Intel X86 architecture). The Register Alias Table (RAT) is used for the mapping.

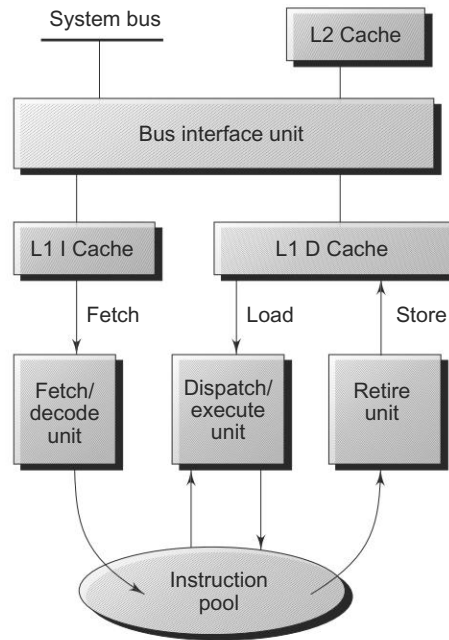


Fig. 13.54 Core engines of Pentium-pro

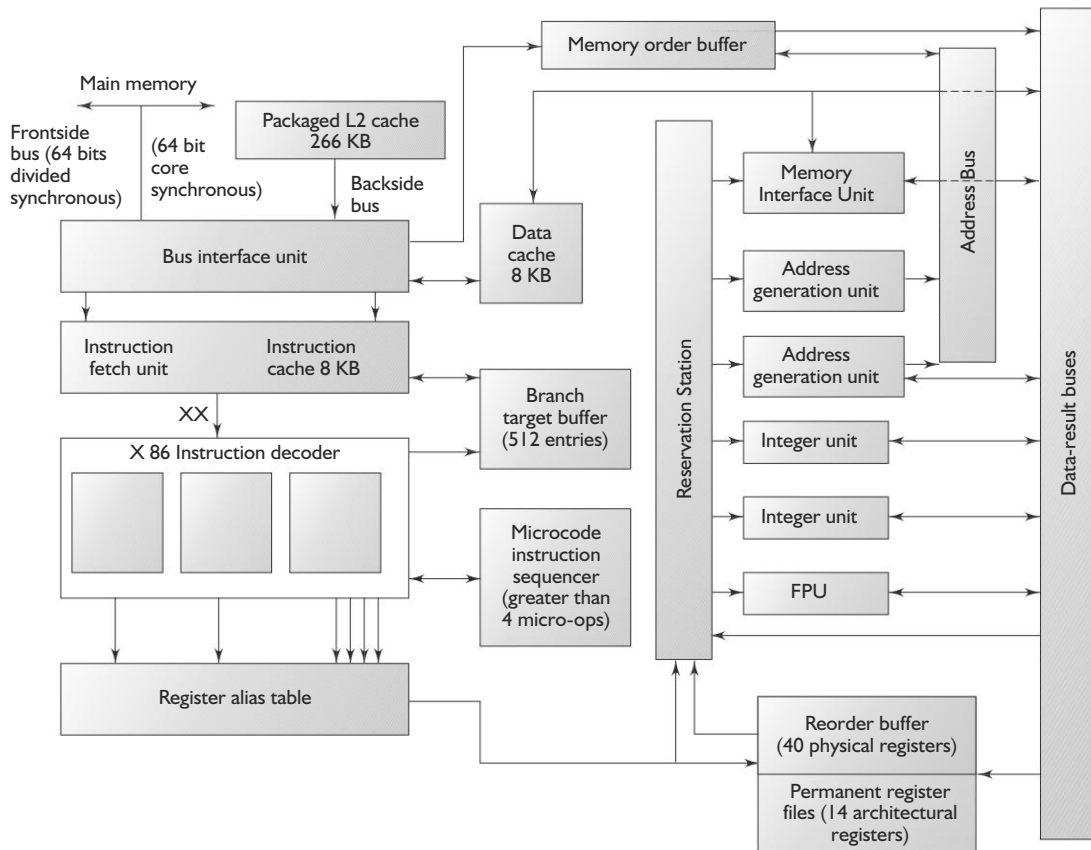
In the case of a branch instruction, the processor speculatively executes the instructions from the predicted branch address even before the branch result is known.

As shown in Fig. 13.55, the Pentium-pro has five execution units in addition to branch unit:

1. Two Integer units
2. Two Address Generation Units (AGU) for load/store operations
3. One Floating point unit

A central reservation station is used.

The Pentium-pro's secondary cache is a static RAM. It is linked to the basic processor by a 64-bit bus, known as the backside bus. The read/write operations with the L2 cache occurs at the internal speed of the processor. The operations on the front side bus occurs at the speed of the system bus. This Dual Independent Bus (DIB) architecture enhances the bandwidth and performance to many folds, in comparison to a single-bus processors.



The Pentium pro fetches instructions from the cache. A decoder converts the instructions into micro-ops and assigns them to Registers in the reorder buffer. The reservation station dispatches the micro-ops to five execution units. After completion, results are committed to the retirement register file and restored to original program order.

Fig. 13.55 Organization of Pentium-pro

13.11.1 Dual Independent Bus (DIB)

The DIB architecture (Fig. 13.56) includes level 2 cache in a dedicated, high speed cache bus allowing the system bus to be free from the cache traffic. It enhances the system bandwidth to many folds and also provides a dramatic improvement in the system performance and scalability.

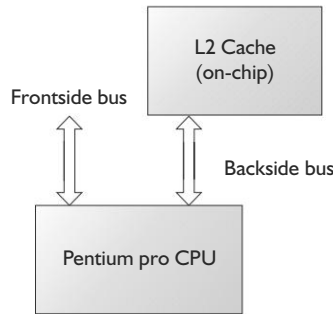


Fig. 13.56 Dual bus architecture

The Pentium II and pentium 3 processors use the dynamic execution technology of Pentium-pro which consists of the following three different facilities:

1. *Multiple Branch Prediction*: It predicts program execution through several branches, accelerating the flow of work to the processor.
2. *Dataflow Analysis*: It creates an optimized, reordered schedule of instructions by analyzing data dependencies between the instructions.
3. *Speculative Execution*: It carries out instructions speculatively thereby ensuring that the multiple execution units remain busy, boosting the overall performance.

13.11.2 Transactional Buffer

The frontside bus in the Pentium-pro is a transactional I/O bus. The Pentium-pro supports a Memory Order Buffer (MOB) which queues upto eight transactions for the frontside bus. Even if the memory access initiated by the Pentium-pro is not complete, it can begin with another bus cycle. The MOB is used to record the information about the in-progress transactions (bus cycle). Eight such outstanding transactions can be registered in the MOB. Hence, the Pentium-pro keeps on initiating a new bus cycle while executing several non-dependant instructions. The control logic coordinates the housekeeping operations, when a previously outstanding transaction reaches the completion stage. Posting appropriate status, deleting the MOB entry and updating the relevant registers are the actions normally performed in such a situation.

13.11.3 Instruction Processing

The instruction decoder divides each CISC instruction into multiple micro-ops. The objective is simultaneous dispatch and parallel execution of the micro-ops of multiple instructions.

Pentium-pro has a 14-stage superpipeline. The latency for each of these pipeline stage is shorter than that of the Pentium's pipeline stage. The actions performed by the Pentium-pro pipeline stages are as follows:

Stage 1

- Calculation of the next value of the instruction pointer (equivalent to PC).

Stages 2–4

- Fetch two cache lines of 32 bytes each and mark the instruction boundaries and pass the 16 aligned bytes to the decoder.

Stages 5 and 6

- Two simple decoders and one complex decoder
- Simple decoders convert complex instructions into micro-operations. Complex instruction is handed over to the complex decoder by a simple decoder. An extremely complex instruction is handled by the microcode instruction sequencer.

Stages 7 and 8

- These two stages continue and finish the task of preparing the micro-ops for superscalar issue.

In stage 7, register renaming is achieved. References to logical registers are reassigned to the physical registers by the Register Alias Table (RAT). There are 40 extra GPRs besides the eight integer and eight floating-point GPRs provided by the architecture. Due to register renaming, at any instant, any of the extra physical register can represent the logical register specified by architecture.

In stage 8, additional status information is stored in the registers. The expanded register file is used as a general purpose instruction pool called the reorder buffer (ROB).

The ROB is a 40-entry of content addressable memory (CAM) arranged as a circular FIFO buffer. The micro-ops are held in ROB during the various stages of execution. The status bits indicate the state of each micro-op and also the execution unit, which can handle that type of micro-op.

13.11.4 Out-of-Order Execution

If an instruction execution depends on the result of a previous instruction whose execution is not complete, the Pentium-pro decides to execute the instruction in a non-sequential

order. Subject to the dependency problem, the Pentium-pro can execute instructions in any order. Eventually, the CPU stores the results in an order which the programmer desires. The objective of the out-of-order execution is effective utilization of the CPU's hardware resources. Along with the branch prediction and speculative execution, the out-of-order execution enables the CPU to process the instructions in an order that optimizes the loading of a CPU's internal resources.

The Pentium-pro supports a reservation station unit to manage the out-of-order execution. It controls the scheduling order in which the micro-ops are dispatched from the ROB to the multiple execution units.

13.11.5 Speculative Execution

Like the Pentium, the Pentium-pro also predicts the outcome of the branch instruction and begins the instruction fetching based on it. Besides, the Pentium-pro speculatively executes these instructions even before the result of branch is known.

In one clock cycle, the reservation station can dispatch upto five micro-ops to the execution units. By checking the status bits of the micro-ops waiting in the ROB, the reservation station schedules dispatching. A micro-op can be dispatched for execution provided the following conditions are fulfilled:

1. There is no dependency constraint
2. Operands are ready
3. An appropriate execution unit is available

13.11.6 Execution

For a complex micro-op, the execution consists of many stages. Similar to Pentium, the FPU in Pentium-pro consist of separate hardware units for multiply, divide and shift operations. The load operation in Pentium-pro requires only one micro-op, as it requires only the starting memory address and the width of the data. A store operation must generate the memory address and data. Thus, it is divided into two micro-ops by the instruction decoder. Generally, these can be executed in parallel.

13.11.7 Completion

On complete execution of the micro-op, its status flag is modified and stored in the ROB. The retire unit constantly scans the ROB for such micro-ops. On tracing a completed micro-op, the retire unit verifies the validity of its retirement. The retire unit places the micro-ops back into the original program sequence and analyzes the interrupts, faults, breakpoints

and mispredicted branches. An interrupt makes the results of micro-ops that follow the interrupted micro-op as useless. After being sure of these special effects, the retire unit stores the results. The copying of result from physical register to appropriate logical register is carried out at this stage. The logical register set is known as Retirement Register File (RRF).

13.11.8 Branch Prediction

When a branch instruction enters the ROB, extra status bits are added to it. These indicate a predicted target address and a fall-through address. The Branch Target Buffer (BTB) supports 512 entries and store the actual target addresses. It follows a 4-bit dynamic history algorithm. By using this, the BTB keeps track of the outcome of the prediction. The Pentium-pro's branch prediction is more than 90% accurate.

13.12 NetBurst Architecture and Pentium 4

The NetBurst Architecture is used in the contemporary Pentium 4 model, which is a three-way superscalar processor. Figure 13.57 shows the internal architecture of Pentium 4 processor. The significant features of the Pentium 4 are briefly described below:

(i) Hyper Pipelined Technology

NetBurst pipeline has 20 stages, which enhances the performance of the processor significantly.

(ii) Level 1 Execution Trace Cache

The branch prediction is better in the NetBurst as the level 1 instruction cache has been replaced by a trace cache. Instead of caching instructions, it caches micro-ops. Besides, it identifies the traces i.e. the sequence of instructions that cross conditional branches. The current versions can cache about 12 K decoded micro-ops during the program execution, and the actual cache size is 20 KB. This delivers a set of instructions to the processor's execution unit and a quick recovery from the branches which are mispredicted.

13.12.1 Rapid Execution Engine

An additional integer and address computation unit are present in the Pentium 4. The integer units operate at twice the speed of a core processor clock. For example, in a 2.4 GHz Pentium 4, the integer ALUs operate at 4.8 GHz. This allows the execution of the basic integer instructions such as Add, Subtract, Logical AND, Logical OR, etc. in half of a clock cycle.

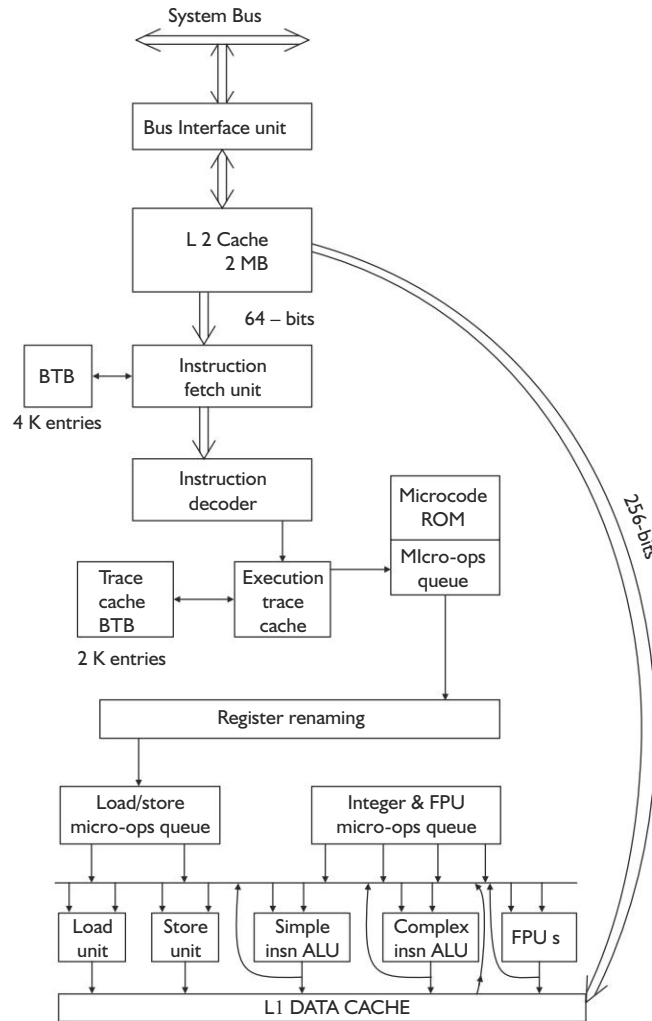


Fig. 13.57 Internal architecture of pentium-4 processor (Not all the functional units are shown)

13.12.2 256 KB, Level 2 Advanced Transfer Cache

The level 2 Advanced Transfer Cache (ATC) delivers a higher data throughput channel between the level 2 cache and the processor core. The advanced transfer cache consists of a 256-bit (32-byte) interface that transfers the data on each core clock. As a result, the Pentium 4 processor (1.50 GHz) can deliver the data transfer at a rate of 48 GB/s.

13.12.3 Advanced Dynamic Execution

The advance dynamic execution engine is a deep, out-of-order speculative execution engine which engages the execution units. The Pentium 4 processor can view 126 instructions in flight, and handle upto 48 loads and 24 stores in the pipeline. It also includes an enhanced branch prediction algorithm which reduces the number of branch mis-predictions by 33% over the Pentium-pro processor's branch prediction capability. It supports a 4 KB branch target buffer that stores the details about the history of past branches, besides implementing a more advanced branch prediction algorithm.

13.12.4 Register Renaming

The Pentium-pro's reorder buffer has been replaced by the register renaming scheme in Pentium 4. Also, there are more registers available (128 vs. 40 in the Pentium-pro reorder buffer).

13.12.5 Internet Streaming SIMD Extensions 2 (SSE2)

With the introduction of SSE2, there are 144 MMX instructions. These instructions include 128-bit SIMD integer arithmetic, and 128-bit SIMD double-architecture which enhances the SIMD capabilities (MMX technology and SSE technology delivered), by adding 144 new precision floating-point operations. These new instructions enhance the overall performance, by reducing the program size. This is significant for several applications including the video, speech, image, photo processing, encryption, financial, engineering and scientific applications.

13.12.6 Data Prefetch Logic

This anticipates the data required by an application program, and also does advance loading into the advanced transfer cache, thereby increasing its performance.

13.13 The Alpha Family

Alpha: It is the Digital Equipment Corporation's (DEC) successor to VAX series of processors. DEC was purchased by Compaq who have subsequently merged with HP. The VAX is a typical CISC processor. Alpha is the first 64-bit architecture. The Alpha 21064 is two-way superscalar, and has four functional units. The integer (2 units) and load/store pipelines consumes seven clock cycles whereas the FP unit consumes 10 clock cycles. It has on-

chip 8 KB instruction and data caches. The instruction length is 32 bits, and the addresses are 64 bits. The 21164 is a four-way superscalar though it has only four functional units. Besides the data cache and instruction caches of 8 KB each, an on-chip 96 KB level 2 cache is also present. The FPU pipeline requires nine clock cycles. The 21264 has six functional units but no on-board level 2 cache. However, the level 1 caches become more complex with 64 K byte each.

The 21364 has on-chip memory controller. The 21464 was planned to be a single core processor with multithreading, but it was cancelled.

13.13.1 Special Features in Alpha

There are some special operations such as scaled add/subtract which rapidly pass through multi-dimensional arrays without any address computation. There is 128-bit multiply instruction. However, there is no integer divide operation present in hardware. The support for the previous VAX floating-point formats as well as IEEE standard floating-point are also provided. As a standard practice, the floating-point exceptions are imprecise; thus, for an arithmetic exception (interrupt), it is not possible to recover a precise architectural state. In some special cases, the programmer can make it precise (for debugging and standards compliance), by placing a special instruction called TRAPB after every floating-point operation. The TRAPB temporarily stalls the processor until all the previous arithmetic instructions are guaranteed so as not to cause exceptions. However, this reduces the performance significantly.

The cancelled 21464 (EV8) was supposed to be with 8 integer and 4 floating-point pipelines and 3MB S-cache. It was to support the SMT (Simultaneous Multi-Threading), providing a concurrent execution of up to 4 software streams inside a single core.

The Alpha instruction set is a simplified one to suite pipelining. It consists of 5 groups:

- integer instructions;
- floating-point instructions;
- branching and comparison instructions;
- load and store instructions;
- PALcode instructions.

Integer division was by emulation.



13.14 PowerPC

The PowerPC is a collaboration project by Apple, IBM and Motorola. It is an enhancement of IBM RISC architectures, with influences by Motorola's 8800 RISC processors. PowerPC supports a low-cost RISC architecture. The first PowerPC processor was the 601 developed

in 1993. It had a clock rate of 60–120 MHz, and a unified 32 Kbyte level 1 cache along with 2.8 million transistors. The 601 speculatively executed the instructions. The 604 supports the split instructions and data caches (16 Kbyte each, 32 Kbyte in the 604e). The current versions are the 750 (G4) series and the 740 (G3) series with clock rates of 200–466 MHz and 350 MHz – 1.25 GHz, respectively. G4s also consists of AltiVec i.e. a set of vector instructions and registers intended to support multimedia-type operations (similar to Intel's MMX and SSE). PowerPC processors are currently manufactured by IBM and Motorola.

PowerPC 620 has 6 functional units with 3 Integer units, 1 Floating point unit, 1 load/store unit and 1 branch execution unit. It uses register renaming and out-of-order execution.

PowerPC has been renamed as Power ISA. Initially planned for personal computers, PowerPC processors have become popular as embedded and high-performance processors also. PowerPC is based on IBM's earlier POWER architecture.

13.14.1 Special Features in PowerPC

The architectural set up of the PowerPC mostly follow the standard RISC architecture except for the Register 0. An exception to this is that if you use a register 0 as the base register in memory address computations, then it is zero. There is a special link register to handle the return addresses for procedure/function calls. This speeds up the procedure call/returns. Another special register (count register) with autodecrement feature, is used as a loop index for the loops. This speeds up the conditional branches using the register.



13.15 SPARC

The SPARC (Scalable Processor ARChitecture) initially developed by Sun Micro Systems is currently an open IEEE-standard architecture, maintained by the SPARC International Consortium. Sun Micro System is the contemporary manufacturer of the UltraSPARC-series processors. The UltraSPARC III has a clock rate of 1.2 GHz, 64 Kbyte level 1 instruction cache and 32 Kbyte level 1 data cache. This is followed by UltraSPARC IV and UltraSPARC V processors. SPARC follows the early RISC architecture. For example, it supports a register window technique comprising of many delayed and one non-delayed unconditional branch instruction. SPARC also consist of multimedia-support instructions called the VIS.

The SPARC architecture is fully open and non-proprietary. The complete design of Sun Micro Systems' UltraSPARC T1 microprocessor was released-in open-source form, and it was named the OpenSPARC T1. The SPARC processor technology is freely available. For example, the SPARC design has been used by Fujitsu Laboratories Ltd. to create the

processor Venus SPARC64 VIIIfx which is capable of 128 billion floating point operations per second (128 GFLOPs).

SUMMARY

RISC processors have a limited number of simple instructions. Though it results in longer programs, the instruction execution is easier and faster. The processors have very little hardware and provides better performance than the CISC processors. Instruction pipelining and on-chip cache memory helped the RISC processor to complete one instruction for every clock cycle.

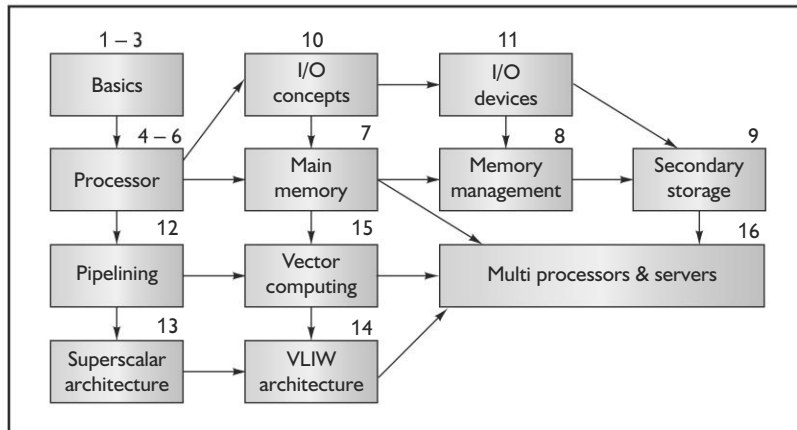
Two recent developments in parallelism of a uni-processor are the *superscalar processor* and *VLIW processor*. These are scalar processors that are capable of executing more than one instruction in each cycle following two different strategies. The superscalar architecture allows execution of two or more successive instructions simultaneously in different pipelines. As a result, more than one instruction gets completed in every clock cycle. Superscalar architecture uses multiple execution units to enable the processing of more than one instruction at a time; it is a uni-processor architecture that can execute two or more scalar operations in parallel. Often the functional units remain idle, unless the superscalar processor uses special techniques such as speculative execution, branch prediction, out-of-order execution, register renaming, write buffers etc.

Loop unrolling is a special technique that converts loop level parallelism into instruction level parallelism. In dynamic superscalar processors, instruction scheduling is done by the processor hardware. In static superscalar processors, the compiler organises the parallelism in the object code by scheduling at compile time.

CHAPTER 14



VLIW AND EPIC ARCHITECTURE



Chapter Flow Diagram

OBJECTIVES

- ◀◀ *VLIW architecture*
- ◀◀ *EPIC architecture*
- ◀◀ *Itanium processors*

KEYWORDS

VLIW

EPIC

Plan of Execution

Static Scheduling

Trace Scheduling

Tree Instruction

Group

Bundle

Predication

14.1 Introduction

Exploiting ILP by hardware approach was the focus of Chapter 13, which dealt with the superscalar processors which detect parallelism present in the object code dynamically (during program execution). An alternate approach depends on software to discover parallelism in a program statically at compilation time. Though majority of superscalar processors follow dynamic scheduling, some superscalar processors, mostly in embedded applications, follow static scheduling by the compiler. Software pipelining, Loop unrolling, and Static branch prediction are common techniques used by the compiler, for static scheduling.

VLIW processor follows static scheduling but has a different architecture. The compiler combines several instructions, into one large instruction, for simultaneous execution. The EPIC architecture is a modified VLIW architecture. This chapter deals with the concepts and techniques followed in VLIW and EPIC architecture.

14.2 VLIW Architecture

Very long instruction word architecture exploits the Instruction-Level Parallelism (ILP) in programs, by simultaneously executing more than one basic instruction. The VLIW architecture follows static scheduling. The compiler translates high level language program into basic operations that the processor can execute simultaneously. The compiler groups several operations into a very long instruction word. The processor supports multiple functional units to execute many operations within one clock cycle. While executing the instruction, the processor performs the operations in parallel in the appropriate functional units. Each instruction consists of several smaller fields, each of which encode an operation for a particular functional unit. The width of the very long instructions is typically between 128 and 1024 bits. Fig.14.1 shows the block diagram of a VLIW processor.

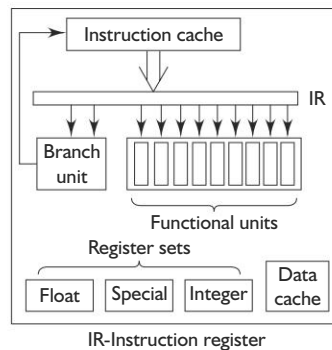


Fig. 14.1 VLIW processor-block diagram

The VLIW offers a strictly defined plan i.e. the Plan Of Execution (POE) created statically during the compilation. The object code determines the following:

1. When each operation has to be executed
2. Which functional units have to be used
3. Which registers contain the operands

A VLIW processor consists of a set of functional units (adders, multipliers, branch units, etc.), apart from the registers and the cache memory. With a thorough knowledge about the organization of a processor, the VLIW compiler creates a POE. The compiler delivers the POE, via the instruction set, to the hardware that implements the POE.

The VLIW processor fetches a very-long instruction word containing several basic operations, and dispatches it for parallel execution. The processor's control logic is very simple, as it does not perform any dynamic scheduling (reordering of instructions) unlike the superscalar processors.

The VLIW rectifies the drawbacks of a superscalar architecture. Due to increase in the number of functional units, the instruction scheduling hardware in the superscalar processor becomes complex. The practical limit of a superscalar design is around five or six instructions dispatched per cycle.

In a VLIW architecture, the software carries out the scheduling. The compiler analyzes the program, selects all the instructions without dependencies and joins them as very long instructions. The VLIW processor shown in Fig.14.2 can execute upto eight operations per clock.

14.3 VLIW Compiler Techniques

The compiler packs many basic operations into a single *instruction word* to load the multiple functional units at a time. For achieving this, the compiler must *create* sufficient Instruction Level Parallelism (ILP) in the object code. The compiler uses multiple techniques such as speculative scheduling of the instructions across the basic blocks, software pipelining etc. The compiler identifies all the data dependencies, and resolve them by rearranging the entire program by moving relevant blocks of the code up and down.

14.3.1 Static Scheduling

In statically-scheduled systems, the compiler identifies and schedules all the instruction-level parallelism in a program by making the parallelism explicit in the instruction set architecture. This eliminates the run-time scheduling issues. The compiler explicitly specifies the following in the object code:

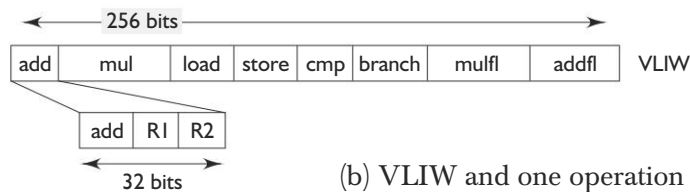
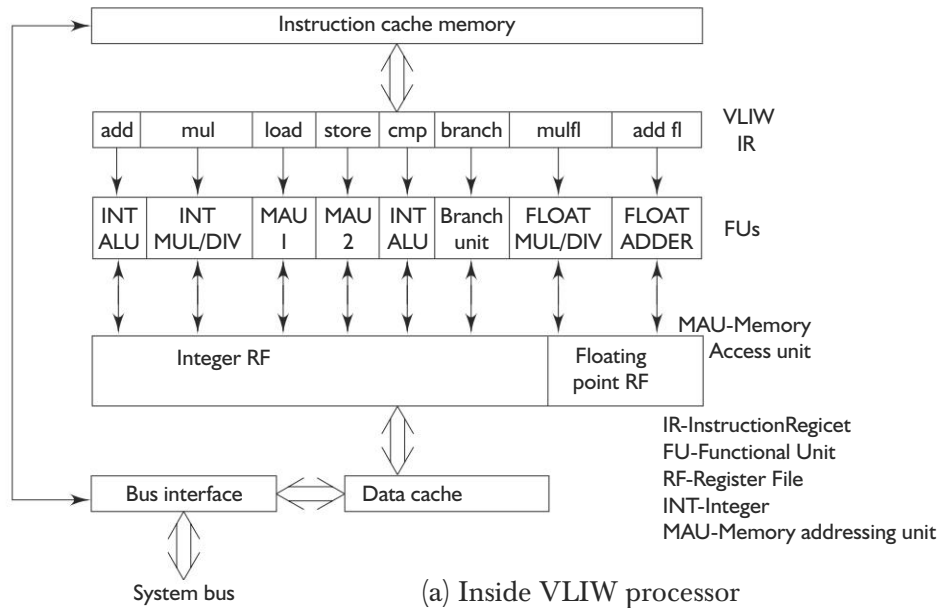


Fig. 14.2 VLIW processor organization

1. Which instructions have storage conflicts with anti- and output-data dependencies.
2. Which instructions are issued and executed in parallel.

The compiler faces an infinite *instruction window*, and uses the overall program knowledge, program semantics (dependencies) and resource constraints in constructing each instruction schedule. The instruction alignment is not an issue in static scheduling, as the compiler schedules instructions in the fetch blocks.

The delayed branch is a software technique for eliminating the delay due to conditional branch instructions. The instruction following the branch instruction acquires a delay slot. The delay-branch schedulers perform limited movement of the instructions across the basic block boundaries. The compiler can move the instructions down from the basic block to fill the branch delay slot. The compiler tries to rearrange the instructions, wherever possible (without affecting program logic), so that some unrelated instructions are moved after the conditional branch instruction that can be continued in the pipeline. The branch does not depend upon these instructions. Hence, it is not important whether the processor executes them before or after the branch. Other ways to fill branch delay slots is to lift an instruction up from the fall-through (or the target basic block). These instructions should not have any side effects, so that their execution does not affect the processor state, if the branch changes the direction. Though these schemes can fill a single branch slot effectively, its effectiveness reduces as the number of branch slots increases. For instance, a compiler can fill approximately 70% of single branch slots, but only 25% of the double branch slots.

14.3.2 Trace Scheduling

Trace scheduling is a compiler technique that enhances the compiler's ability to move instructions, across basic block boundaries. It utilizes branch predictions to *create* a single large block of code, from individual basic blocks. A greater level of concurrency is achieved by scheduling this large block, instead of the individual basic blocks. A trace is a possible path through a program—the route program execution takes for some set of input data. A trace scheduling compiler optimizes at the level of whole traces instead of the basic blocks. The compiler uses information gathered by analysing the program, choosing the most likely trace, and scheduling it as a big basic block. This mechanism is repeated for all branch results. The superscalar RISC such as Pentium-pro speculatively executes some of the instructions, whereas a VLIW compiler moves up these instructions *before* the predicted branch. However, these can be restored *later* if needed.

The compiler uses information which is collected by profiling the program. The compiler predicts the most suitable route, and plans the pathway as one big basic block, and later, repeats the procedure for all other program branches. The VLIW compiler moves the code up or down till it detects branching (according to the tracing). A VLIW processor can offer special support to the compiler. Two such cases are:

1. A multiway branch operation allows several branches to be packed in a single wide instruction, and the processor performs it in a single clock cycle.

2. Conditionally executed operations, whose execution depends on the results of the previous operation, can replace many explicit software branches altogether.



14.4 VLIW vs Superscalar Processor

Though the VLIW processor, consumes less power and provide high performance than superscalar processor chip, it suffers from the following demerits:

1. The object code size is very large due to aggressive scheduling policies followed by the compiler. Hence, the object code requires more memory.
2. The compilers are smart and powerful. The VLIW compiler development is an involved and time consuming process; therefore it is expensive.
3. In view of exhaustive tasks performed by the VLIW compiler, the compilation is a slow process.
4. A VLIW compiler needs an in-depth knowledge of the hardware details of its processor, including the number of functional units and their individual latencies.
5. A new compiler is developed for a new VLIW processor. A new version of the VLIW processor cannot recognize the object code of an old VLIW processor; the old software needs recompilation. One indirect solution to this issue is to divide the compilation process into two stages:
 - (a) All the software must be prepared in the hardware-independent format using an intermediate code.
 - (b) The intermediate code has to be translated into the processor-dependent code during installation on the end-user hardware.
6. The VLIW is inefficient for the object-oriented and event-driven programs. Can a program perform properly during the dynamic run-time events (such as waiting for I/O) unforeseen at the compilation time? But, the out-of-order RISC processor easily tackles it.

MultiFlow, Culler and Cydrome are some of the old VLIW computers of 1980s. The Intel i860 is the first VLIW processor based on a single chip¹. FPS computers (AP-120B, AP-190L and some of the later models) and Kertsev's M10 and M13 and Elbrus-3 are popular VLIW computers. Intel and HP have a joint plan on modifying the VLIW architecture called EPIC. Intel's Itanium is based on the EPIC.

¹The Intel i860 is sometimes considered as a RISC processor rather than a VLIW processor.

Table 14.1 compares the superscalar and VLIW architectures.

TABLE 14.1 Superscalar and VLIW architectures

Sl. no.	Item/feature	Superscalar architecture	VLIW architecture
1	Instruction size	Small/medium length	Very long; fixed length; 128 to 1024 bits
2	Parallelism in object code	Hidden; The problem of revealing parallelism must be handled at hardware level; the hardware develops an action plan for revealing the hidden parallelism	Explicit and revealed by compiler; the program gives precise data on parallelism — the compiler reveals parallelism in the program and notifies the hardware
3	Compiler-hardware interaction	Loose; compiler does not know about details of hardware/functional units	Tight; compiler knows about details of hardware functional units
4	Instruction scheduling and parallel dispatch	Dynamic scheduling by processor during run time	Static scheduling by compiler during compilation time
5	CPU complexity	Complex; special hardware for instruction scheduling and dispatch	Simple
6	Compiler complexity	Simple	Complex
7	Branch prediction	Hardware method by processor	Compiler detects from the source code
8	Instruction execution; POE	Processor develops POE dynamically	Compiler develops and supplies a fixed POE to be obeyed by the processor
9	Extent of parallelism	Within a basic block	Across basic blocks
10	Memory requirement for object code	Normal	Very large
11	Compilation time	Short	Long

(*Contd.*)

Sl. no.	Item/feature	Superscalar architecture	VLIW architecture
12	Downward compatibility (object code level) to older CPUs	Easy to provide	Normally impossible; may need dynamic recompilation or use of intermediate code
13	Maximum number of instructions per clock cycle	5 to 6	8 and higher
14	Suitability for event driven programs	Good	Inefficient
15	Special techniques	Out of order execution and speculative execution	Trace scheduling
16	Inspiration/early generation	IBM System/360 model 91, CDC 6600, CRAY-1 etc.	Concept of horizontal microinstruction applied at a higher level
17	Special effort by high level language programs	Nil	Nil

14.5 Tree Instruction

The VLIW approach exploits the instruction-level parallelism in a branch-intensive programs. The IBM approach is based on expressing a program as a sequence of tree-instructions or trees, each of which contains a multi-way branch and multiple operations, all executable concurrently. Each *tree* (Fig. 14.3) corresponds to an unlimited multi-way branch with multiple branch targets and an unlimited set of primitive operations. All operations and branches are independent, and executable in parallel.

The multi-way branch is associated with the internal nodes of the tree, whereas the operations are associated with the arcs. The multi-way branch is the result of a set of binary tests on condition codes; the left outgoing arc from a tree node corresponds to the false outcome of the associated test, whereas the right outgoing arc corresponds to its true outcome. Based on the evaluation of the multi-way branch, a single path within a tree-instruction is selected at the execution time, as the *taken path*. The operations on the taken path are

completely executed, and their results are placed in the corresponding target registers or memory locations. In contrast, the operations on the discarded path of the multi-way branch are inhibited from sending their results to the memory or the registers, which does not produce any effect on the processor.

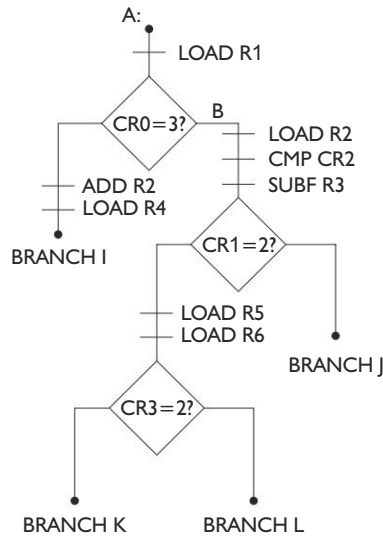


Fig. 14.3 Tree instruction



14.6 EPIC Architecture

The Explicitly Parallel Instruction Computing (EPIC) architecture has evolved from the VLIW architecture, while retaining many concepts of the superscalar architecture. The EPIC approach extracts the *hidden parallelism* from the instruction level, using the Wide Issue Width (WIW) and Deep Pipeline Latency (DPL). It follows two key aspects of the VLIW:

1. The processor does not check for dependencies between operations since these are already identified by the compiler as independent entities.
2. The processor does not have the out-of-order execution logic since the instruction issuing is responsibility of the compiler.

The compiler cannot resolve the entire ambiguity that can only be removed during the execution, for which the processor should have dynamic mechanisms. The EPIC supports

these mechanisms at the architecture level, so that the compiler can control the dynamic mechanisms using them selectively. This is also essential since the EPIC architecture aims to provide some amount of downward compatibility to the Intel's earlier processors such as Pentium 4.

14.6.1 IA64 Architecture

The IA64 supports a EPIC architecture developed jointly by Intel and HP. It follows speculative instruction execution, and usage of predicates. The first IA64 processor is the Itanium. Its architecture is similar to the HP-PA RISC architecture.

The IA64 is a load/store architecture with 64-bit memory addresses and registers. The register files are as follows:

1. 128 general-purpose registers of 64 bits
2. 128 floating point registers of 82 bits
3. 64 predicate registers of 1 bit
4. 8 branch registers of 64 bits (hold branch destination addresses)

The itanium has a RISC 2- style register stack, with some deviations. The first 32 registers are global and the remaining are available for procedures/functions. Instead of overlapping the register windows, the calling procedure's first local register is used as a pointer to the called procedure's local registers, for passing the parameters.

14.6.1.1 Groups and Bundles

The instructions are divided into groups and bundles. A bundle is a group of three instructions packed into a 128-bit word. It includes three 41 bit fields for instructions and one 5 bit template slot. The instructions of the bundle are executed in parallel by different functional units. The possible interdependencies, which prevent parallel execution of the instructions from the same bundle, are identified in the *template* field.

A *group* is a sequence of instructions that are executed in parallel provided the following two conditions are fulfilled:

1. Availability of sufficient hardware.
2. No Memory Dependencies. Due to static scheduling, the compiler guarantees that no data dependencies occur between the registers, but the processor must control the dependencies in the memory.

There is no limit for the group length. But the boundary between the groups is explicitly marked by a *stop*. A stop is inserted between the instructions within a bundle. The stop marks the end of a group of instructions that can be executed in parallel.

Apart from indicating the presence of the stops, the template indicates the types of instructions in a bundle. The template contains the compiler supplied information about how instructions can be executed in parallel. The options are:

- I — ALU and non-ALU Integer. These include the integer arithmetic/logical operations, together with the various moves, tests and shifts.
- M —ALU and Memory. These consists of the integer arithmetic/logical operations along with the memory access.
- F —Floating point. This comprises of the floating-point operations.
- B —Branches. This contain the static prediction information which is used till the branch prediction algorithm builds-up enough history information.
- L + X—Extended.

For example, a template value of 0 indicates three instructions containing the M, I and I instructions, respectively, with no stops, whereas a template value of 1 indicates the same sequence, but with a stop at the end. On the other hand, a template value of 10 indicates the sequence M, M and I, with a stop between the two Ms.

The 41 bits in the instructions provide the following details:

1. The first 4 bits indicate the major opcode.
2. Some bits are used to precisely indicate the operation.
3. The final 6 bits specify the predicate register.
4. The remaining bits specify operands.

Three 7-bit register fields are used for addressing the 128 general purpose registers or 128 floating point registers.

14.6.1.2 Predication

Predication is a way of dealing with the conditional branches (instead of flushing the pipeline or prediction). Each instruction is conditional. It is executed only when a specific condition or predicate is true. There is a set of 64 1-bit predicate registers. The condition of a predicate flag decides whether an instruction should be executed or not. Actually the instruction is executed in the pipeline, but its result is written in the destination, only if the predicate flag is 1. The comparison instructions are used to set and clear the predicate registers, and the subsequent instructions are conditionally executed, depending on their values. For example, consider the following source code:

```
if  $a = b$  then  
     $a = 0$ 
```


else
 $b = 0$.

A possible predicated version is as follows:

```
CMPEQ Ra,Rb,P1/P2
[P1]MOV Ra, 0
[P2]MOV Rb, 0
```

The CMPEQ instruction compares a and b . If they are equal, the predicate register P1 is set to *true*; otherwise it is set to *false*. The predicate register P2 is set to the inverse of P1. The next two instructions are each predicated on one of P1 and P2. The first MOV sets Ra to zero if P1 is true; otherwise, it just becomes a no-op. Thus, there is no need to predict the outcome of a branch. In Itanium, most of the instructions are predicated, with the 6 bit predicate field selecting one of the 64 predicate registers.

14.6.2 Itanium Processors

With downward compatibility, the Itanium can execute old code (compiled for the earlier processors such as Pentium 4), without much performance loss. Hence, it has some hardware (similar to superscalar processors) to resolve the dependencies, rename registers, etc.

The Itanium 1 supports a level-3 cache on a separate chip, packaged with the main processor. There are 10 pipeline stages, and the pipeline divides into four parts. The first part deals with fetching and branch prediction (3 stages); the second part manages the instruction issue and register renaming (2 stages); the third part covers the operand fetch (2 stages); and the fourth part handles the execution and write-back (3 stages). It comprises of nine functional units: two I-units; two M-units; three B-units and two F-units. All these are internally pipelined. In the Itanium 2, the level-3 cache is integrated to the main processor. The pipeline is short. Its future version supports a HyperThreading technique. The HyperThreading is a better way of using the under-utilized functional units. It interleaves multiple threads simultaneously. The hardware behaves as a multiple virtual processors.

SUMMARY

The VLIW rectifies the weakness in the superscalar architecture. The instruction scheduling hardware in the superscalar processor becomes increasingly complex with the increase in the number of functional units. In VLIW architecture, the software does the scheduling. The VLIW is an instruction-set philosophy in which the compiler and processor are designed for

better parallelism with less hardware in the processor. The compiler packs several operations into a very long instruction word. The processor has multiple functional units to execute many operations within one clock cycle. Each instruction contains several small fields, each of which encodes an operation for a particular functional unit. While executing the instruction, the processor issues the operations in parallel to the appropriate functional units. The practical limit of superscalar design is around five or six instructions dispatched per cycle. A typical VLIW processor can execute around eight operations per clock.

The EPIC architecture retains two key aspects of VLIW:

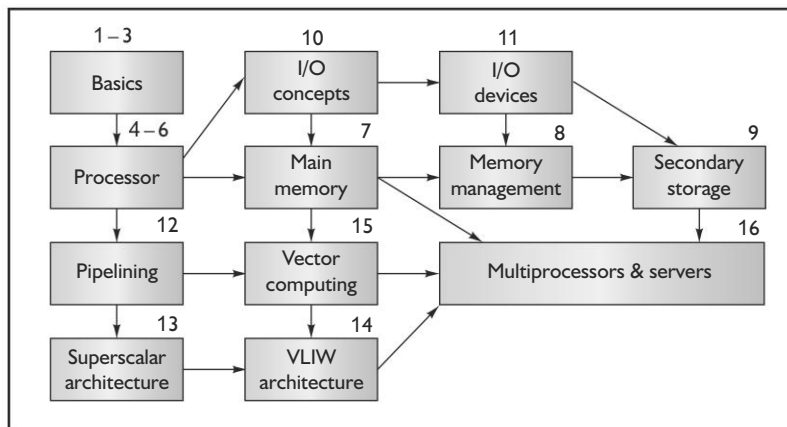
1. The processor does not check for dependencies between operations since these are already identified by the compiler as independent.
2. The processor does not have the out-of-order execution logic since the instruction is issued by the compiler.

The EPIC architecture aims to provide some amount of downward compatibility to the earlier processors. The compiler cannot resolve the ambiguity, which can only be removed during execution, for which the processor should have dynamic mechanisms. The EPIC supports these mechanisms at an architecture level so that the compiler can control the dynamic mechanism using them, selectively.

CHAPTER 15



VECTOR COMPUTING AND ARRAY PROCESSING



Chapter Flow Diagram

OBJECTIVES

- ◀◀ *Vector processing*
- ◀◀ *Array processing*

KEYWORDS

Vector

Arithmetic Pipeline

Vector Register

Scalar Register

Chaining

Scatter Operation

Gather Operation

Address Pipeline

Dedicated Array Processor

Attached Array Processor

 15.1 Introduction

Vector computing and array processing are two popular techniques of achieving parallelism to get high performance. These techniques have been in use mostly in supercomputers and special purpose systems. However, similar techniques are also used in some recent microprocessors for multimedia applications. This chapter reviews the basics of vector computing and array processing.

 15.2 Vector Computing

A vector processor is used for high-performance scientific computing, where matrix and vector arithmetic are common. Some examples are:

- Weather forecasting
- Space flight simulations
- Image processing
- Remote sensing

These are some of the applications that perform some basic set of operations repeatedly on a large amount of data. The traditional processors used for such computations took several days. However, this type of applications is easily vectorizable and executable on a vector processor. The Cray Y-MP and the Convex C3880 are two examples of vector processors used nowadays.

A vector processor efficiently handles arithmetic operations on elements of arrays, known as the *vectors*. For a single vector instruction, the vector processor simultaneously performs multiple homogeneous operations on an array of elements. The vector processor uses data parallelism, by compact parallel datapath structures, with multiple functional units operating for the same vector instruction. Since a single instruction specifies a whole set of operations, time spent on instruction fetch and decode operations is reduced compared to the traditional processor which needs a series of instructions.

15.2.1 Vector Arithmetic

A vector, v , is a list of elements

$$v = (v_1, v_2, v_3, \dots, v_n),$$

where n is the number of elements in the vector. In a computer program, the vector is an array of one dimension. The terms vector, array and list are used interchangeably. For example, in Fortran, we declare v by the following statement:

$$\text{Dimension} = v(N)$$

where N is an integer variable indicating the *length of the vector*.

Arithmetic operations can be performed on vectors. Two vectors are added by adding the corresponding elements:

$$\mathbf{Z} = \mathbf{x} + \mathbf{y} = x_1 + y_1, x_2 + y_2, \dots, x_n + y_n.$$

In Fortran, vector addition is specified by the following routine:

```
DO 300 I = 1, N
  300 Z(I) = x(I) + y(I)
```

where $\mathbf{Z}$ is the vector for the sum and $\mathbf{Z}$, $\mathbf{x}$, and $\mathbf{y}$ are known as the arrays of dimension N . This operation is called *elementwise* addition. For traditional processor (scalar), the compiler translates the previous program into the following sequence of instructions (furnished as assembly language statements):

```
INITIALISE I = 0
300 READ x( I )
   READ y( I )
   READ N
   ADD Z(I) = x( I ) + y( I )
   INCREMENT I = I + 1
   IF I ≤ N GO TO 300
CONTINUE
```

While executing the program, the scalar processor performs the addition instruction, n times, in order to add n elements of array x , with corresponding elements of array y . The loop is encountered n times. If n is more, time spent on instruction fetch and decoding is significant. In the case of the vector processor, a single vector instruction covers all the n items as indicated previously.

15.2.2 Vector Computing Concepts

Vector processing has similarities with SIMD model of parallelism. In vector processors, operations are performed on vectors or linear array of numbers. While executing a vector instruction, a vector processor uses the pipelines to operate on a few vector elements, during each clock cycle. On the other hand, the SIMD processor operates on all the elements at the same time.

A vector processor contains a set of arithmetic *pipelines*. These pipelines overlap the execution of the different parts of an arithmetic operation, on the elements of the vector. This helps in efficient execution of the arithmetic operation. Let us examine how a vector pipeline operates.

15.2.2.1 Floating-point Operation

Consider the following steps for a floating-point addition on a traditional processor:
 $Z = x + y$.

1. *Exponent Comparison*: The exponents of the two floating-point numbers are compared to find the number with the smaller magnitude.
2. *Mantissa Alignment*: The mantissa of the number with the smaller magnitude is shifted so that the exponents of the two numbers match.
3. *Mantissa Addition*: The mantissas are added.
4. *Normalization*: The result of the addition is normalized.
5. *Exception Testing*: Checking is done to detect if any floating-point exception such as overflow occurred during the addition.
6. *Result Rounding*: Rounding off the result is done.

Figure 15.1 presents the step-by-step progress of such a scalar addition. The numbers considered are $x = 1328.00$ and $y = -742.6$. For simplicity sake, these are represented in decimal notation with a mantissa of four digits.

Step →	1	2	3	4	5	6
x	0.1328E4	0.13280E4				
y	-0.7426E3	-0.07426E4				
z ($x+y$)			0.058540E4	0.58540E3	0.58540E3	0.58540E3

Fig. 15.1 Six stages of floating-point addition

Suppose this addition has to be performed on all the elements of a pair of vectors (arrays) of length n . All the six stages are executed for every pair of elements. If each stage of the execution takes δ units of time, then each addition takes 6δ units of time (excluding the time taken to fetch and decode the instruction or fetching the two operands). Hence, the time taken to add the elements of the two vectors in a serial fashion is $T_s = 6n\delta$. Figure 15.2 displays the execution stages with respect to time.

15.2.2.2 Arithmetic Pipeline

Suppose the floating-point addition operation is pipelined by six different arithmetic sections, then each stage of the addition is performed at each section in the pipeline. Each

section of the pipeline consists of a separate arithmetic unit designed for the operation to be performed at that stage. Once stage 1 is completed for the first pair of elements, the output is transferred to the next section 2, while the second pair of elements enters into the first section 1. Assume each section takes δ units of time. The data flow through the pipeline stages with respect to time is shown in Fig. 15.3.

Time	δ	2δ	3δ	4δ	5δ	6δ	7δ	8δ
Step								
1	x_1+y_1						x_2+y_2	
2		x_1+y_1						x_2+y_2
3			x_1+y_1					
4				x_1+y_1				
5					x_1+y_1			
6						x_1+y_1		

Fig. 15.2 Scalar floating-point addition of vectors

It is obvious that it takes same amount of 6δ units of time to obtain the sum of the first pair of elements. But, the sum of every successive pair is available at intervals of δ units of time. Hence the time, T_p , to do the pipelined addition of two vectors of length n is

$$T_p = 6\delta + (n - 1)\delta = (n + 5)\delta$$

Time	δ	2δ	3δ	4δ	5δ	6δ	7δ	8δ	9δ	10δ	11δ
Step											
1	x_1+y_1	x_2+y_2	x_3+y_3	x_4+y_4	x_5+y_5	x_6+y_6	x_7+y_7	x_8+y_8	x_9+y_9	$x_{10}+y_{10}$	$x_{11}+y_{11}$
2		x_1+y_1	x_2+y_2	x_3+y_3	x_4+y_4	x_5+y_5	x_6+y_6	x_7+y_7	x_8+y_8	x_9+y_9	$x_{10}+y_{10}$
3			x_1+y_1	x_2+y_2	x_3+y_3	x_4+y_4	x_5+y_5	x_6+y_6	x_7+y_7	x_8+y_8	x_9+y_9
4				x_1+y_1	x_2+y_2	x_3+y_3	x_4+y_4	x_5+y_5	x_6+y_6	x_7+y_7	x_8+y_8
5					x_1+y_1	x_2+y_2	x_3+y_3	x_4+y_4	x_5+y_5	x_6+y_6	x_7+y_7
6						x_1+y_1	x_2+y_2	x_3+y_3	x_4+y_4	x_5+y_5	x_6+y_6

Fig. 15.3 Pipelined floating-point addition of vectors

The first 6δ units of time includes the time to fill the pipeline and the time to obtain the first result. After receiving the last result, the pipeline is flushed.

The pipeline mode of the addition is faster than the serial mode by nearly the number of stages in the pipeline. If n is large, the pipeline mode addition is about six times faster than scalar addition.

The number of stages in a floating-point addition differs in different processors—more or less than six. The operations for floating-point multiplication are slightly different from the operations for addition; also the number of sections in a multiplication pipeline is different from an addition pipeline.

15.2.2.3 Vector Registers

Most vector processors have *Vector registers*. In comparison to a general purpose or a floating-point register that stores a single value, a vector register stores several elements of a vector, simultaneously. The contents of a vector register are transferred to the vector pipeline, one element at a time.

15.2.2.4 Scalar Registers

A *Scalar register*, like general purpose or floating-point registers, stores a single value. But, it is configured to be used by a vector pipeline. The value in the register is read once every δ units of time and released into the pipeline, which is similar to receiving a vector element from the vector pipeline. This facilitates the operation on the elements of a vector by a scalar. For example, consider the computation:

$$j = 3.4 \times i$$

The constant 3.4 is stored in a scalar register, and transmitted into the vector multiplication pipeline, every δ units of time, in order to participate in the multiplication of each element of i .

15.2.2.5 Chaining

Generally vector processors have multiple pipelines which are of different types. In some vector processors, the output of one pipeline is directly released into another pipeline. This technique is called *chaining*. It eliminates the intermediates storage for the result of the first pipeline before transmitting it into the second pipeline. Figure 15.4 displays the use of chaining in the computation of following vector operation:

$$K(\mathbf{x} + \mathbf{y})$$

where $\mathbf{x}$ and $\mathbf{y}$ are vectors and K is a scalar constant. Chaining doubles the number of floating-point operations that are performed in δ units of time. Once both the multiplication and addition pipelines are filled, two floating-point operations (one multiplication and one addition) are completed every δ time units. It is possible to chain more than two functional units, but it involves complex timing considerations.

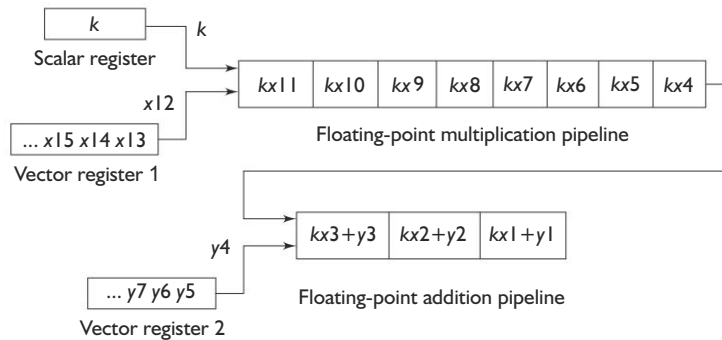


Fig. 15.4 Chaining to compute $k\mathbf{x} + \mathbf{y}$

15.2.2.6 Scatter and Gather Operations

Suppose only certain elements of a vector are required for computation. The vector processor picks up the appropriate elements (a *Gather* operation), and joins into a vector in a vector register. If the elements used are in a regularly-spaced pattern, then the spacing between the elements to be gathered is called the *stride*. For example, if the elements:

$$\mathbf{x}1, \mathbf{x}6, \mathbf{x}11, \mathbf{x}16, \dots,$$

are to be extracted from the vector:

$$(\mathbf{x}1, \mathbf{x}2, \mathbf{x}3, \mathbf{x}4, \mathbf{x}5, \mathbf{x}6, \dots, \mathbf{x}n)$$

for some vector operation, the stride is 5. A *Scatter* operation reformats the output vector so that the elements are spaced correctly.

15.2.2.7 Vector-Register Vector Processors

A vector-register vector processor fills the vector pipelines from the vector element values currently in the vector registers. This reduces the time to fill the pipelines (the startup time) for vector arithmetic operations; the vector registers are filled while the pipelines are performing some other operation. The vector results are put back into a vector register after

the completion of the operation, or they may be piped directly into another pipeline for an additional vector operation (chaining).

In these processors, arithmetic or logical vector operations are performed only on vectors that are already in the vector registers. A *load vector* operation reads the elements of the vector from the memory into the vector register. The vector result of a vector operation is stored in a vector register. It is stored in the memory by a *store vector* operation. This operation can be overlapped with other operations. For subsequent requirements of the result vector, it is read from the vector register.

15.2.2.8 Memory–Memory Vector Processors

The memory–memory vector processor has no vector registers. It fetches vectors directly from memory to fill the pipelines, and stores the pipeline results directly to memory. The startup time for the vector pipelines, is more because it takes more time, to start a vector operation due to increased memory access time. One example of a memory–memory vector processor is the CDC Cyber 205.

Due to reduced vector access time and the overlap between store vector and other operations, vector–register vector processors are usually more efficient than memory–memory vector processors. However, if the vectors are long, the difference in efficiency is insignificant. On the other hand, the memory–memory vector processors are attractive if the vectors are extremely long.

15.2.2.9 Interleaved Memory Banks

To reduce the access time for vector elements stored in memory, the memory of a vector processor is usually interleaved into multiple *memory banks*. Successive memory banks possess successive memory addresses cyclically. In a k way interleaved memory, there are k number of memory banks. The word $\bar{0}$ is stored in bank $\bar{0}$, word $\bar{1}$ is in bank $\bar{1}$, ..., word $k - \bar{1}$ is in bank $k - \bar{1}$, word k is in bank $\bar{0}$, word $k + \bar{1}$ is in bank $\bar{1}$, ..., etc. When the elements of a vector are read from the interleaved memory, the reads are staggered across the memory banks, so that one vector element is read from a bank per clock cycle. If one memory access takes n clock cycles, then n elements of a vector may be fetched in one memory access; this is n times faster than the same number of memory accesses to a single bank.

15.2.3 Vector Computing Performance

For most vector processors, the time to complete one pipeline stage (δ) is equivalent to one clock cycle of the processor. (In some processors, it is equal to two or more clock cycles.) Once a pipeline is filled, it generates one result for every δ units of time, that is, for each clock cycle. In other words, the hardware performs one floating-point operation per clock cycle.

Let k be the number of δ time units needed by the same sequential operation (or the number of stages in the pipeline). Then the time to execute that sequential operation on a vector of length n is:

$$T_s = kn\delta$$

and the time to perform the pipeline mode is

$$T_p = k\delta + (n - 1)\delta = (n + k - 1)\delta$$

As before, for $n > 1$, $T_s > T_p$.

Startup time: The startup time is the time needed to initiate the operation. In the traditional sequential machine, there are overheads to set up a loop to repeat the same floating-point operation for an entire vector. Also, the elements of the vector are fetched from memory. If S_s is the number of δ time units for the sequential startup time, then T_s is presented as:

$$T_s = (S_s + kn)\delta$$

In a pipelined processor, the flow from the vector registers or the memory to the pipeline has to initiate; this time is S_p . The overhead cost for $k\delta$ time units is the time required to initially fill the pipeline. Hence, T_p must include the startup time for the pipelined operation; thus,

$$T_p = (S_p + k)\delta + (n - 1)\delta$$

or

$$T_p = (S_p + k + n - 1)\delta$$

Since, the length of the vector is very large, (as n goes to infinity), the startup time becomes negligible in both cases. Hence,

$$T_s \sim kn\delta$$

while

$$T_p \sim n\delta$$

Thus, for large n , T_s is k times larger than T_p .

Table 15.1 lists some typical vector computers. Most of the supercomputers are vector computers which offer very high performance for special applications. However, they are not suitable for standard general purpose applications.

TABLE 15.1 Comparison of vector computers

Vector computer	Type	Additional remarks
CDC Star 100	A supercomputer with built-in vector processor	Based on APL programming language
TI ASC	Suitable for long vectors	—
CDC Cyber 205	A memory-memory vector processor	Has four general-purpose pipelines; it also provides both gather and scatter operations
Alliant FX/8	A shared-memory multiprocessor with eight CPU's, each with an attached vector processor	—
Cray-1	Supercomputer with pipelined vector arithmetic units; A vector-register processor	Provides scatter and gather operations; uses chaining, has 12 different pipelines or <i>functional units</i> ; each vector register contains 64 single elements; each element or word contains 64 bits
Cray X-MP	Shared-memory multiprocessor with each CPU controlling its own set of vector processors; more support for overlapped operations along with multiple memory pipelines	Chaining of all three floating-point pipelines is allowed
Cray-2	A multiprocessor with up to four processors	Does not support chaining

15.2.4 CASE STUDY 3: Cray-1

Cray-1 is a supercomputer with pipelined vector arithmetic units. It is a vector-register processor. Each vector register stores 64 elements. Each element or word has 64 bits. The Cray-1 has 12 functional pipelined units. All the 12 units can operate concurrently. The Cray-1 is the first vector processor to apply chaining technique. These are four groups of functional units:

- *Vector pipelines* that perform integer or logical operations on vectors.
- *Floating-point pipelines* that execute floating-point operations using scalars or vectors.
- *Scalar pipelines* that carry out integer or logical operations on scalars.
- *Address pipelines* that perform address calculations.

Figure 15.5 illustrates the functional units and registers. There is no floating-point divide unit. A floating-point reciprocal approximation pipeline is used for floating-point division, that is, x/y is computed as $x(1/y)$.

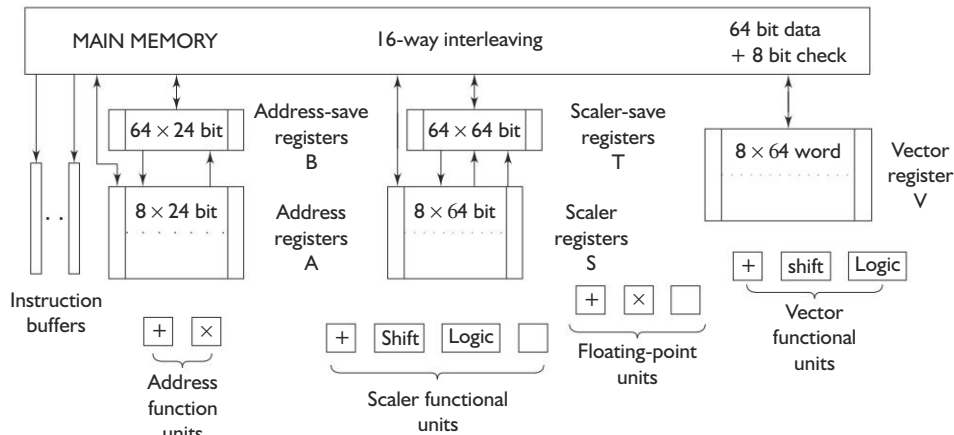


Fig. 15.5 Cray-1 Functional units and registers

There are mainly five types of programmable registers: A, B, S, T and V. In addition, there are some supporting registers: Vector Length register (VL), Vector Mask register (VM), Program counter (P), Base Address register (BA), Limit Address register (LA), Exchange Address register (XA), Flag register (F) and Mode register (M). The 'A' registers are known as address registers. They work as address registers for memory reference, and as index registers. Also, these are used for shift counts and loop counts. The 'B' registers are called as address-save registers, and are used as auxiliary storage (buffer) for the 'A' registers. The 'S' registers are known as scalar registers, and are used as source and destination registers for scalar arithmetic and logical instructions. The 'T' registers are known as scalar-save registers which are used as auxiliary storage for the 'S' registers. The 'V' registers are known as vector registers, and are used as source and destination registers for the vector functional units (pipelines).

The Cray-1 allows one memory read and write per clock cycle. Hence, it can read only one vector, and write one vector result at the same time. When more than one read (or write) is required for the same operation, one of the operations is postponed.

Consider the multiplication of two vectors of length greater than 64:

$$z = x \times y$$

Initially, two vector registers are loaded from memory: one, the first 64 elements of x ; and the other, the first 64 elements of y . The result for z enters a third vector register. After the first 64 elements are covered, the source vector registers are reloaded from memory, and the result vector register contents are stored in memory. Since only one read and one write are executed per cycle, elements of both input vectors cannot be read at the same time; so the pipeline delays one of the read operations.

The main memory is a 16-way interleaved memory. It is a SEmiConDuctor memory with Error Detection and correction (SECDED) logic. There are 12 I/O channels.

The Cray-1's Fortran compiler is an optimizing compiler. The programmers need not modify their old source programs since the compiler takes care of 'vectorization' of DO loops and generates vector instructions.

The Cray-1 is the first of a series of Cray supercomputers.

15.3 Array Processor

An array processor performs simultaneous computations on the elements of a vector (an array or a table of data in multi-dimensions). It executes a sequence of necessary actions, for an operation, on the entire array. The multiple units in the array processor perform the same operation on different data. The common usage of the array processors include analysis of fluid dynamics, and rotation of 3D objects, weather data processing as well as data retrieval, in which the elements of a database are scanned simultaneously, signal processing and applications in which differential equations, matrix manipulations or linear algebra are involved. Often people fail to differentiate between array processor and vector processor. A vector processor does not operate on all the elements of a vector at the same time, it only operates on the multiple elements concurrently. The array processors are classified into two types:

1. Dedicated Array Processor
2. Attached Array Processor

The *Dedicated array processor* is a stand alone computer system, whereas the *Attached array processor* is attached to another general purpose computer system as an extension. ILLIAC IV and STARAN are two popular array processors. Figure 15.6 illustrates the interfacing of an attached array processor to a general computer. The FSP-164/MAX is an attached processor for VAX 11 system. Its attachment (interfacing) to another general purpose computer adds vector architecture to the general purpose computer. The dedicated array processor falls under SIMD architecture. Hence, it is generally referred as SIMD array processor.

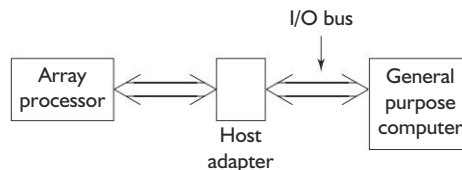


Fig. 15.6 Attaching an array processor

15.3.1 CASE STUDY 4: ILLIAC IV—a SIMD Array Processor

The ILLIAC IV is a special purpose array processor. Its initial design had 256 Processing Elements (PE) organized as four quadrants of 64 PEs. The 64 PE quadrant is an 8 by 8 array of PEs. Each PE is linked to its neighbors in the four directions. Each PE has 2K words of memory. The central Control Unit (CU) broadcasts instructions that are executed by the PEs. The CU is a highly complex and sophisticated unit with its own resources for executing certain scalar instructions. In other words, the CU is equivalent to a processor. Figure 15.7 illustrates the structure of ILLIAC IV. All the PEs perform the instruction operation (say ADD) simultaneously on different components of vectors. The ILLIAC IV is linked to B6500 control computer that acts as the link between the user and ILLIAC IV. Logically, the ILLIAC IV is linked as a peripheral subsystem to B6500.

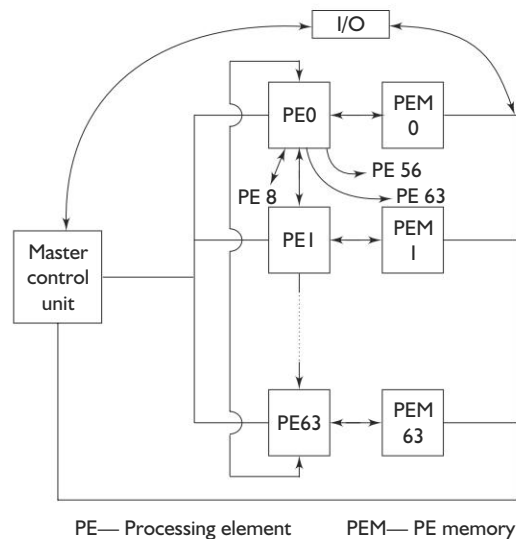


Fig. 15.7 ILLIAC IV array processor

SUMMARY

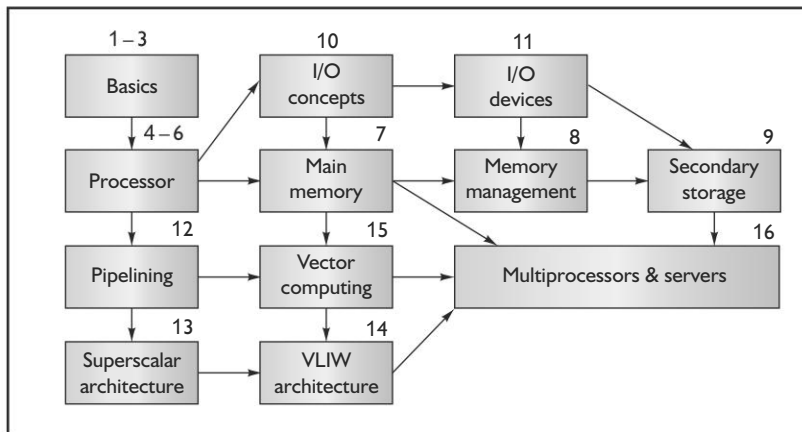
A vector processor efficiently handles arithmetic operations on elements of arrays called vectors. For a single instruction, the vector processor simultaneously performs multiple homogeneous operations on an array of elements. It uses the data parallelism by parallel datapath structures with multiple functional units operating for the same vector instruction. An array processor performs simultaneous computations on the elements of a vector (an

array or table of data in multi-dimensions). It executes a sequence of actions necessary for an operation on an entire array. The multiple units in the array processor perform the same operation on different data. The dedicated array processor is a stand-alone computer system whereas the attached array processor is attached to another general purpose computer system as an extension.

CHAPTER 16



MULTIPROCESSOR SYSTEMS AND SERVERS



Chapter Flow Diagram

OBJECTIVES

- « Multiprocessor architecture
- « Clustering
- « Cache coherency
- « Fault tolerance
- « Server architecture

KEYWORDS

Tightly Coupled Multiprocessor
Loosely Coupled Multiprocessor
Cluster
Crossbar Switch
Multiport Memory
Multistage Switching
Hypercube Interconnection
Global Resources
Local Resources
Shared Memory

Distributed Memory
Cache Coherence
Snooping
Failure
Fault
Graceful Degradation
Availability
Reliability
Serviceability
Bath Tub Curve

 16.1 Introduction

A multi-processor system is a computer system comprising two or more processors. An interconnection network links these processors. The primary objective of a multi-processor system is to enhance the performance by means of parallel processing. It falls under the MIMD architecture as discussed in chapter 12. Beside providing high performance, the multi-processor system also offers the following benefits.

1. *Fault tolerance and graceful degradation:* The system continues after failure at reduced power. This is essential for some critical applications: Defence, Spacecraft control, Airline reservation system, Web server, etc.
2. *Scalability and modular growth:* The system configuration (e.g. the number of processors etc.) is enhanced at the required increments, at any point of time.

Generally, most multiprocessor systems are scalable. The hardware and system software are designed with scalability in mind. The number of processors in a multiprocessor system can be increased at any time, and the system software can be suitably configured. Building a high performance computer system by linking together several low performance computers is a standard technique of achieving parallelism. This idea is the basis for development of multiprocessor systems. Designing a microcomputer using multiple single-chip microprocessors has been a cost-effective strategy for several years in the past. The latest trend is design of multicore microprocessors resulting in quantum change in the way multiprocessor systems are developed and used for various applications.

Server systems are high performance systems dedicated for specific special functions. Apart from providing high performance, server systems offer very high level of reliability. This chapter provides an overview of multiprocessor systems and server systems. In addition, basic concepts of fault tolerance are presented in this chapter.

 16.2 Classification of Multi-processors

Multiprocessor systems differ in the way the following issues are tackled by the hardware and the system software:

1. How does the operating system coordinate with the processors?
2. How do the processors share the common data without any clash or conflict?
3. How do the processors share the common hardware resources?

Proper synchronization has to be maintained so that no processor uses obsolete value of a shared data when another processor is updating the data. There are two popular techniques for this: *locking* and *message passing*. In the lock technique, a processor can access the shared variable only if it is unlocked, meaning that no other processor is using it currently. Any one

processor can acquire the lock at a time. All other processors requiring to access the shared data wait till the original processor unlocks the variable. In the message passing technique, the software of one node sends message (with data) to another node for transmitting the data.

Figure 16.1 illustrates the different types of multi-processors. In a tightly coupled multi-processor, shown in Fig. 16.2, the multiple processors share information via a common memory (global memory). Hence, this type is also known as the shared memory multiprocessor system. In a shared-memory multiprocessor system, all processors share a single memory address space. All processors can access any memory location. The processors communicate among themselves through shared variable in memory. Besides sharing the global memory, each processor can also have local memory dedicated to it which cannot be accessed by other processors in the system. In a loosely coupled multiprocessor system, the memory is not shared, and each processor has its own memory, as shown in Fig. 16.3. This type of a system is known as the distributed memory multi-processor system. Multiple private memories form the distributed memory. The communication among the processors has to be explicit. The information is exchanged by the processors via the interconnection network, by a common message passing protocol.

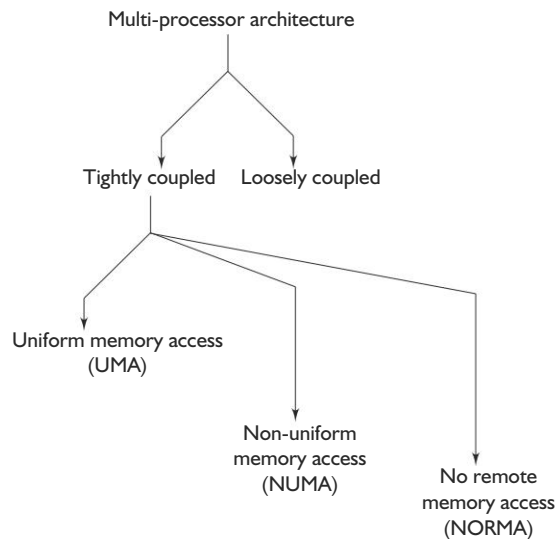


Fig. 16.1 Multi-processor system types

The loosely coupled multi-processor system has physically distributed memories. These Systems are of two types: Distributed Shared Memory (DSM) and cluster. In DSM system, the processors have a common or shared address space for all the memories. It is also known as Non-Uniform Memory Access (NUMA) system. In this, the access time of the memory is different for the processors. Some processors can access memory faster than others. In the cluster system, there is no sharing of address space.

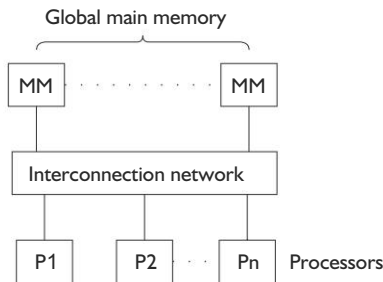


Fig. 16.2 A tightly coupled multi-processor system

In a Uniform Memory Access (UMA) system, the access time of memory is equal for all the processors irrespective of which processor accesses which portion of the common memory. Developing software for the NUMA multi-processor system is more complex than for UMA multi-processor system. However, NUMA multi-processor systems can be programmed to achieve higher performance than UMA multi-processor systems.

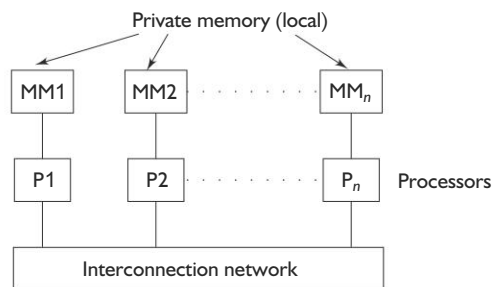


Fig. 16.3 A loosely coupled multi-processor system



16.3 Symmetric Multi-Processor (SMP)

A symmetric multi-processor is an UMA multi-processor system with identical processors, equally capable in performing similar functions in an identical manner. All the processors have equal access time for the memory and I/O resources. For the operating system, all

processors are similar, and any processor can execute it. The terms UMA and SMP are interchangeably used.



16.4 Interconnection Structures

Figure 16.4 illustrates the common types of interconnection structures used in a multi-processor system.

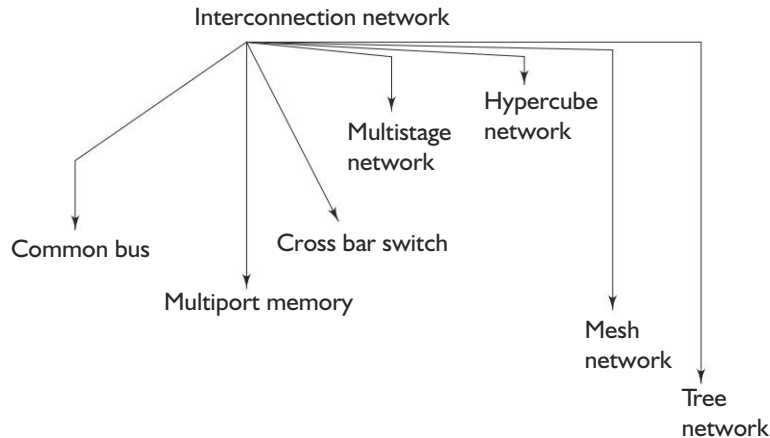


Fig. 16.4 Interconnection structures for multi-processors

16.4.1 Common Bus Structure

The common bus structure is also known as the time-shared bus structure (Fig. 16.5). All the processors, memory and I/O sub-systems are linked through the bus. Besides the common bus, each processor has a local bus (not shown in Fig. 16.5) through which it communicates with the local memory and local I/O.

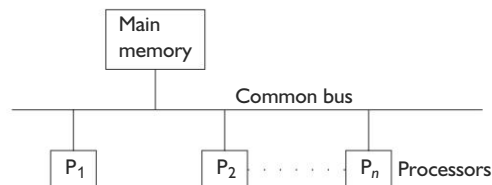


Fig. 16.5 Time-shared common bus

Merit

It is a simple and cheap structure.

Demerit

The communication is very slow as not more than one transfer/communication can occur through the bus at a time. For example, when one processor is accessing the memory, the other processors cannot perform any other operation through the bus.

16.4.2 Multiport Memory Structure

In multiport memory system, there is a separate bus between each memory module and the processor. Figure 16.6 shows a system with n processor and n memory modules. Each memory module has n ports. Each memory module has a priority logic that resolves conflicts occurring due to simultaneous requests from the multiple processors.

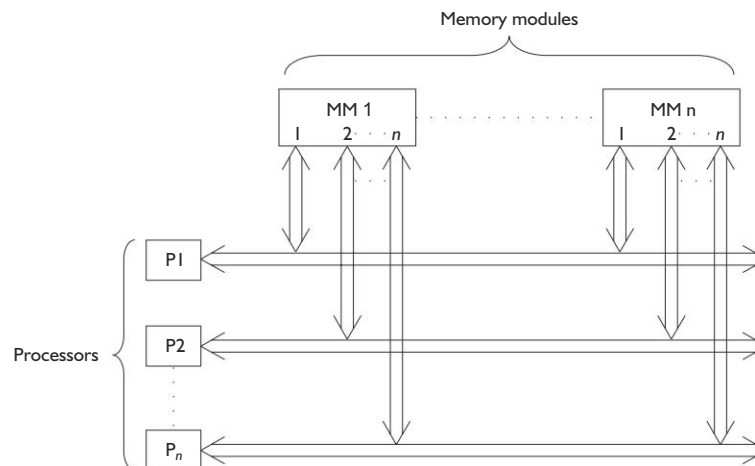


Fig. 16.6 Multiport memory interconnection

Merit

Due to multiple paths, the multiple processors can simultaneously access memory thereby achieving high transfer rate.

Demerit

Large hardware in the memory modules and a lot of interconnecting cables are needed for a large number of processors. Suitable for small systems.

16.4.3 Crossbar Switch

The crossbar switch structure is similar to the crossbar telephone exchange. Fig. 16.7 shows the crossbar switch interconnection for a system with n processors and n memory modules. Each s in the figure is actually an electronic circuit that provides the desired path. It supports priority logic to resolve conflicts.

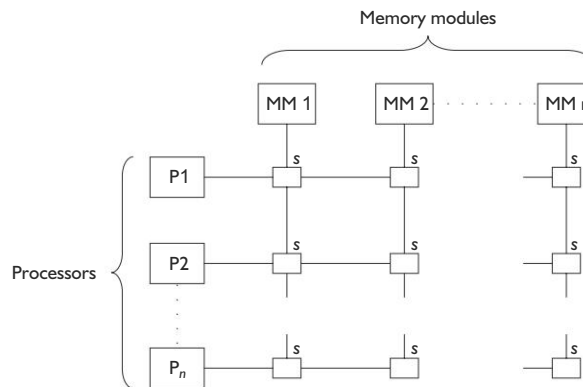


Fig. 16.7 Crossbar switch

Merit

Since each memory module has a separate path, simultaneous access of all the memory modules is possible.

Demerit

A complex hardware is required when a large number of processors are present.

16.4.4 Multistage Switching Network

In multi-stage switch network structure, there are more than one stage of electronic switches (compared to just one stage in crossbar switch structure) used to set-up the paths between the processors and the memory modules. Each switch has two inputs and two outputs. Any input can be connected to any output. Several schemes are possible for interconnec-

tions through the switches. For e.g., the omega switching network is a popular scheme (Fig. 16.8).

Merit

It supports a cost effective structure.

Demerit

Only a limited number of simultaneous communications is possible.

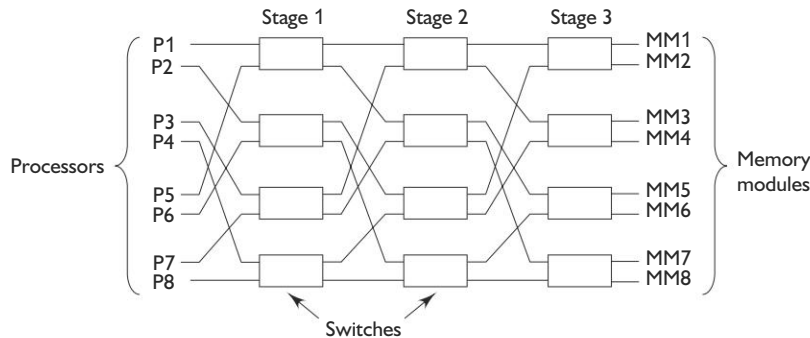


Fig. 16.8 8×8 omega multistage switching network

16.4.5 Hypercube Network

A hypercube is an n -dimensional cube, used to interconnect 2^n processors in a loosely coupled fashion. Each node of the hypercube represents a processor. Figure 16.9 shows a three-dimensional hypercube linking eight processors numbered from 000 to 111. Each node not only acts as a processor but also supports a set of local memory and I/O for the processor. The edges of the cube correspond to the communication paths. Each processor has a dedicated communication paths to its neighboring processors through the edges.

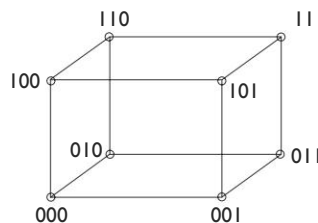


Fig. 16.9 A three-dimensional hypercube

Merit

- (i) Flexibility permits easy scalability to higher configurations by increasing the number of dimensions (n).
- (ii) The intelligent protocols of communication can be implemented easily.

Demerit

Due to the existence of the multiple routes between processors, the complexity of routing increases.

16.5 Clusters

Clustering is an interconnection of the multiple independent computer systems operating as a single system in a collaborative fashion. It is an alternate to the multi-processor system wherein the multiple processors are present in a single computer system. Functionally, a cluster is loosely coupled multiprocessor system without any sharing of address space. Each node (system) in a cluster can also work independently. A cluster is different from a computer network whose main objective is *resource sharing*. A cluster provides all the benefits of a multi-processor system. There are two main objectives behind the formation of a cluster:

1. **Load sharing:** Here, the two systems A and B form a cluster, and they share the processing load (Fig. 16.10). More than two systems also can form a cluster. An increasingly common application of load sharing concept is implementing a web server for a high capacity web site. Multiple computers sharing the load of a web server forming a cluster can handle heavy traffic for the web site.
2. **Fault Tolerance:** In this case (Fig. 16.11), the system B is a hot stand-by for system A. At times of failure in system A, the system B takes over its role. When system A is working properly, the system B silently monitors the normal status of system A by analyzing the *heartbeats* received from it. If the heartbeat is absent or corrupted, system B starts operating.

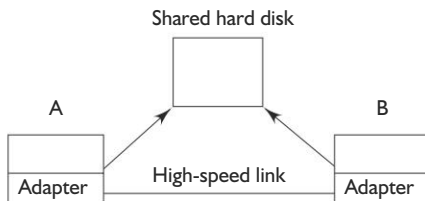


Fig. 16.10 Clustering for load sharing



Fig. 16.11 Clustering for fault tolerance

In both types of clustering, the system software appropriately configures the systems.

 16.6 Cache Coherence Problem

The cache memory is an essential feature of a computer system in order to reduce the impact of the main memory access time on the instruction cycle time. When any processor writes in its cache memory, all processors must get this new value when they need it. They should not receive the old values present in their cache memories. Hence when any processor modifies an item in its cache, all other processors should do one of the following two options:

1. Update their cache memories with the new value being written
2. Invalidate their old entries in their cache memories

When any processor encounters *cache miss* during a read operation, all cache controllers should verify if they have latest copy of the required item. The cache controller that has the latest value should supply the data to the cache controller that encountered the cache miss. A few processors follow *write-through* policy, whereas others follow the *write-back* policy. When more than one processor share a main memory location, its copies are present in the cache memories of all the processors. It is important that when one processor writes in its cache memory, proper action is taken so that the old, invalid information in other cache memory are not used by others. Consider two processors A and B, each with separate cache memory. Assume that a memory location 1000 is mapped to the cache memories of both processors, and the value is *s*. Now, if A modifies this to *N* in its cache memory and B is not aware of it, then while reading the memory location 1000, it receives from the cache memory. Obviously, this should not be allowed. *Cache coherence* is defined as a condition in which all the cache lines of a shared memory block contain same information at any point of time. Cache coherence problem occurs when a processor modifies its cache without maintaining uniformity in all the caches. It is a problem faced by any system which allows the multiple processors to access multiple copies of the data. Both the software and hardware methods are available as solution to the cache coherence problem. The software methods are cheaper but slower, whereas hardware methods are faster but costlier.

16.6.1 Software Solution

A simple software solution to the cache coherence problem is based on the compiler analyzing the source program while generating the object code. The compiler identifies the items shared by the processors. It then declares (tags) the *writable* shared items as non-cacheable. Accordingly, all the processors access these information from the main memory. All processors access the main memory both for read and write operation on the shared items as alerted by their cache memory tag lines. This is a simple and cheap technique as there is no hardware, and the solution is achieved at the compile time. But it is an inefficient solution as the main memory traffic increases during the execution. An alternative method makes the items non-shareable *temporarily* during the critical moments, by a special instruc-

tion inserted by the compiler. It requires an intensive research by the compiler for locating the *safe* and *critical* periods for the shared variables.

16.6.2 Hardware Solutions

The commonly used two different types of hardware solutions to a cache coherence problem are:

1. Cache snooping protocol
2. Directory protocol

16.6.2.1 Cache Snooping Protocol

In this technique, every processor has a snoopy cache controller that constantly monitors the transactions on the bus by other processors. Thus, every processor keeps track of the other processor's memory writes. All cache controllers snoop (monitor) on the bus, to find out whether any processor is modifying a shared item, to ensure that all cache controllers have updated copy of the shared item. In other words, snooping technique ensures that no processor uses outdated (obsolete) value present in its cache memory. Figure 16.12 gives a block diagram for the snooping technique based cache coherence. Two common methods based on snooping are followed: write update protocol and write invalidate protocol.

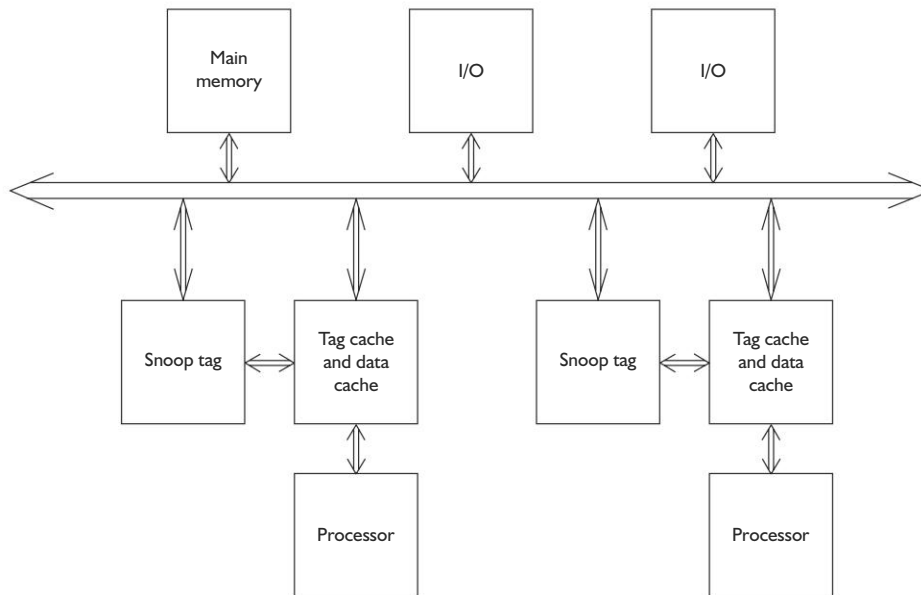


Fig. 16.12 Cache coherence by snooping technique

In write update protocol, when a processor updates a shared item in the cache, it alerts all other processors, and supplies the necessary information (broadcasts the new value) so that other processors can update their caches immediately. The *writing processor* broadcasts the new data through the bus. All other processors update their cache memories with the new value. This scheme consumes more memory bandwidth. By keeping track of the shared words, the extent of broadcasting can be reduced.

In a write invalidate protocol, when processor A writes in its cache (say, for location 1000), it broadcasts the information to all other processors through the bus. The *writing processor* performs an invalidation sequence through the bus. All processors who have mapped the location 1000 in their cache, mark it invalid. Subsequently, if another processor accesses 1000 again, it results in a cache miss and the following action take place:

- (a) In the case of a write-through cache, the entry should have been already updated, by A.
- (b) In the case of a write-back cache, processor A must detect the read request by processor B, and supply the latest value, from its cache.

16.6.2.2 Directory Protocol

This scheme provides a centralized solution by maintaining a directory in the main memory. A single directory keeps the states of all blocks of main memory. The directory includes following information about each block: which cache memories have copies of the block, and the state of the blocks (valid or invalid). Directory entries can be distributed. This directory contains information about the different cache memories in the processors. When a processor modifies the information in its cache, the central memory controller checks the directory and finds which processors are affected. The directory controller sends explicit information to all processors that have the copies of the shared data. Only these affected processors take appropriate action. Fig. 16.13 gives a block diagram for the directory technique based cache coherence.

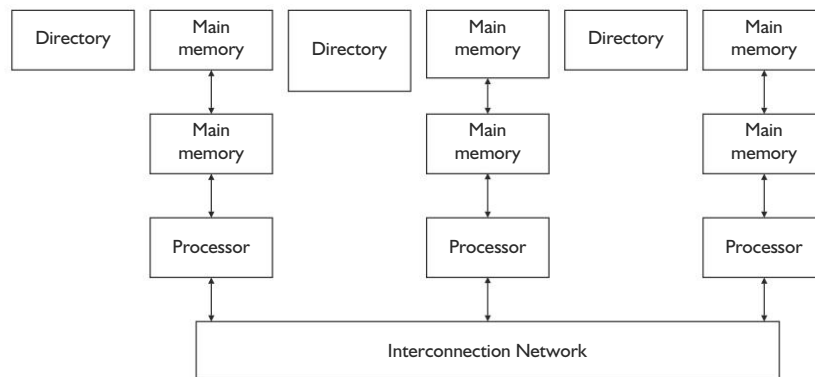


Fig. 16.13 Cache coherence by directory technique

16.6.3 MESI Protocol

The MESI cache coherency protocol is a standard protocol followed by several processors such as Intel Pentium 4 and the PowerPC. In this scheme, each cache line can have four states:

Modified (M): The cache line is modified (*dirty*) i.e. it is not same as in the main memory.

Exclusive (E): The cache line is *clean* and not present in any other cache memory.

Shared (S): The cache line is *clean* but may be present in some cache also.

Invalid (I): The cache line does not contain a valid data.

Two bits in the tag indicate the current state out of the four states.

16.7 Fault Tolerance

The computer failures causes a variety of damages varying from loss of life to destruction of the property since the computers are used everywhere in all spheres of human life. Even winning a war between countries is dependent on the proper functioning of a computer. While nothing can prevent a failure, it is possible to design a computer that can function correctly even with a fault, without causing any damage. *Fault tolerance* is the ability of a computer to execute its operation correctly without being affected by the hardware or software failures. To make a system fault-tolerant, special hardware and/or software modules are added in its architecture. Simple fault-tolerant techniques are present in low cost computers. Some of these are:

1. Error detection/correction using parity check, hamming code, cyclic redundancy check etc. so that the failures during data transfer and data storage is detected, and reported.
2. Replication of hardware module (such as ALU) to perform a function in parallel by two (or more) modules and matching the results.
3. Using three identical modules (such as processor) to perform a function and choosing the result of *majority* (ruling out failure of more than one module simultaneously).

Special fault-tolerant computers are designed with several redundant hardware modules and alternate paths, so that they can function without causing damage. However, their performance level is reduced after a fault, compared to the normal level. This is known as the *graceful degradation* since the fault-tolerant computer works correctly (but with reduced efficiency) even after a failure. Tandem's computers are excellent examples of the fault-tolerant computers.

Though the terms used in fault tolerance computing are common terms found in other fields, the meanings differ sometimes. The standard definition for the frequently used terms are as follows:

Failure: It refers to the abnormal operation/behavior of a component or sub-system in a computer.

Fault: It is a defect or mistake which can occur either in the hardware or software component/subsystem. This can be a design defect or a manufacturing defect. It can be a solid (permanent) fault, intermittent fault or transient fault.

Solid Fault: It is a permanent fault which occurs in a computer when it misbehaves consistently. The failure remains till the fault is rectified.

Intermittent Fault: In this case, the computer's behavior is inconsistent. Sometimes works, sometimes fails.

Transient Fault: A transient fault is induced by an outside environment i.e. electromagnetic interference (EMI), AC voltage fluctuations etc.

Error: It is a noticeable sign/evidence of a fault.

Availability: It is the percentage of a trouble-free operation time of the system (total uptime) to the total time period under consideration. For example, a 99.9% availability indicates a down time of 8 hours per year.

Reliability: It is the probability that the system will survive (function properly) for a period of time. It depends on the reliability of the individual components used in the system. A computer is ready for shipment, from the factory, after the various stages of manufacturing are completed. In a typical computer factory, a standard pre-shipment *burn-in* testing is performed for about a week prior to shipment as a final stage. The finished product (integrated and tested computer system) is kept inside a 'burn-in chamber' and operated there continuously for about 168 hours. The temperature inside the computer is usually maintained at about 70°C. When the computer is cooked like this, the components with poor infant mortality fail. These components are replaced and the burn-in period of 168 hours is restarted all over again. Until no failure is detected during seven consecutive days of burn-in, this process is repeated. It has been found that problems which are likely to appear during the initial 25 weeks of field life are caught in the factory during the *repeated* 168 hours of burn-in. Thus, a computer manufactured under such strict pre-shipment burn-in procedure starts straight away its trouble-free field life of 25 years as shown in the bath tub curve in Fig. 16.14. During this period of 25 years, problems are rare. After this period,

problems occur due to wear and tear. Many computer manufactures often compromise in the pre-shipment burn-in duration due to tight production schedules. Hence the system starts giving problems in the customer site from an early date.

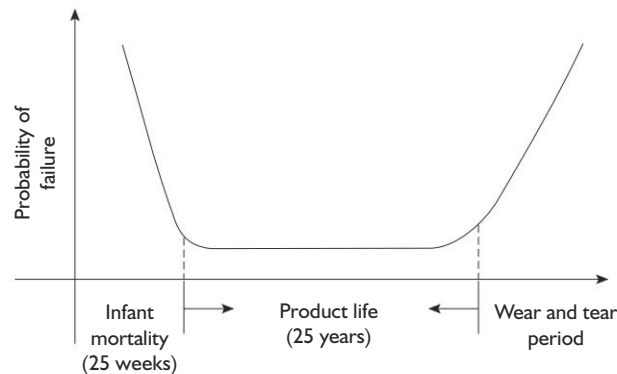


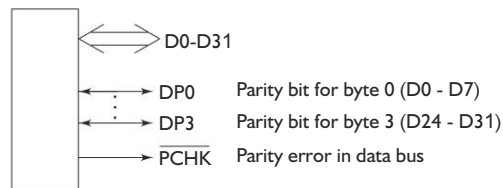
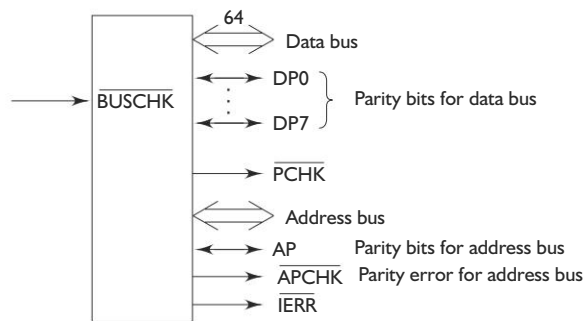
Fig. 16.14 Bath tub curve

16.7.1 Types of Faults: Hardware and Software

An operating hardware may fail at any moment, even in the next second. But, software once proved, always works without any failure. Hence, if a computer has a software fault, it is due to incomplete testing of the software. A software error may appear as a hardware error. The hardware faults are of the following types:

1. Mechanical problem
2. Electronic problem
3. Environmental problem
4. Media problem

Fault rectification consists of three steps: fault detection, fault diagnosis and corrective action. A fault-tolerant system follows an automatic process for all the three steps, or for one or two steps and the remaining steps are performed manually by a trained service engineer. The modern processors incorporate several levels of on-chip fault-tolerant measures. Table 16.1 identifies the varying degrees of the fault tolerance measures in Intel processors used in the PC series, see Figs.16.15 and 16.16

**Fig. 16.15** Parity check for data bus in 80486**Fig. 16.16** Error detection in Pentium**TABLE 16.1** Fault Tolerance Measures in Intel Microprocessors

Sl. no.	INTEL processor	Fault tolerance features	Remark
1	8088/8086	Nil	–
2	80286	Shutdown	Processor automatically halts during double fault
3	80386	Same as 80286 plus BIST (Built-In Self Test)	Processor tests its internal hardware, after reset sequence, before starting program execution
4	80486	Same as 80386 plus following: 1. Parity logic for data bus (Fig.16.15)	On-chip parity greenerator and parity checker; $\overline{PCHK}$ output indicates that microprocessor has detected parity error in data bus inputs during read operation

(Contd.)

Sl. no.	INTEL processor	Fault tolerance features	Remark
5	Pentium	2. Tristate test mode	Helps troubleshooting fault
		Same as 80486 plus following: 1. Parity logic for address bus	$\overline{\text{APCHK}}$ indicates that microprocessor has detected a parity error in address bus inputs
		2. Parity logic for all internal transfers within microprocessor	$\overline{\text{IERR}}$ from the master indicates that there is an internal transfer error; in FRC mode, $\overline{\text{IERR}}$ from the checker indicates that master's result is wrong
		3. Machine check	–
		4. FRC	–
		5. Halt input	–

16.7.2 Functional Redundancy Check (FRC)

The objective of an FRC is to arrest any damage (to data, program or results) by malfunctioning of an execution unit. For this purpose, the two Pentium chips should be used in a Master-Checker configuration, see Fig. 16.17. The FRCMC signal configures the

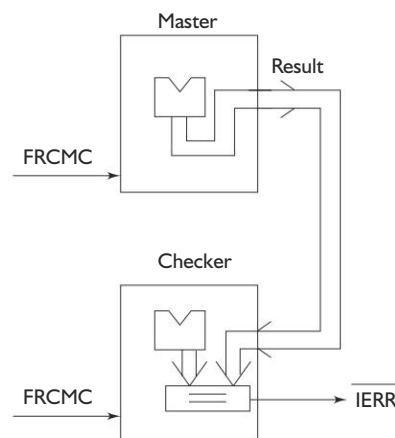


Fig. 16.17 Pentium—Master-checker configuration

processor chip either as a master or as a checker. The execution unit of the checker also performs the arithmetic operations similar to the master's. The master's output are given as input to the checker's corresponding pins which act as input in the checker mode. Both the results are matched in the checker. If there is a mismatch, the $\overline{\text{IERR}}$ signal from the checker turns active. This indicates a fault in the master chip. The system hardware and software cooperate to freeze the program execution, thereby preventing further processing of the wrong result.

16.7.3 Halt Input

The Pentium has a special input pin $\text{R}/\overline{\text{S}}$ which is used, to halt the processor, by the external signal input. As long as the signal is 'HIGH', the processor continues to operate. Once 'LOW,' the processor stops. Also, it generates an output signal PRDY which acts as an acknowledgment for $\text{R}/\overline{\text{S}}$. This facility may be used for any one of the following requirements depending on the system design:

1. Manual halt, through a front panel switch
2. External circuit halt, under abnormal error situation
3. System software halt, through an output port

Figure 16.18 illustrates these options. Depending on the option chosen, the terminal I is connected to one of the input terminals: a, b or c.

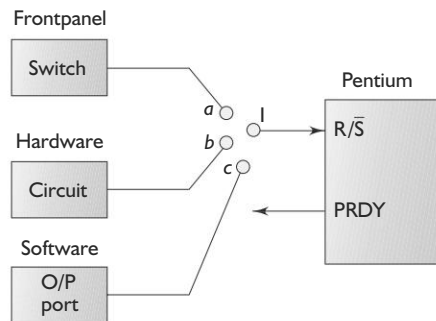


Fig. 16.18 Use of halt input in Pentium

16.7.4 Machine Check

The machine check action is carried out by the Pentium in response to any serious fault encountered in the system. It can be detected either by Pentium or the external hardware. The actions by the Pentium as part of the machine check sequence makes sure that the processor does not continue processing after the abnormal status, thereby preventing unpredictable results. The sequence of actions for the machine check are as follows:

1. Freeze instruction execution
2. Save the internal status in machine check registers (address and control signals)
3. Branch to machine check exception service routine if enabled by the control register

The machine check is performed on the following two occasions:

1. The *external* hardware detects a malfunction in the bus cycle, and hence issues the BUSCHK signal.
2. The Pentium detects *parity error* in the data bus during an input bus cycle (memory read or I/O read). Hence the microprocessor generates the $\overline{\text{PCHK}}$ signal. However, the machine check action, in this situation, can be disabled externally, by making the $\overline{\text{PEN}}$ input signal high.



16.8 Server Systems

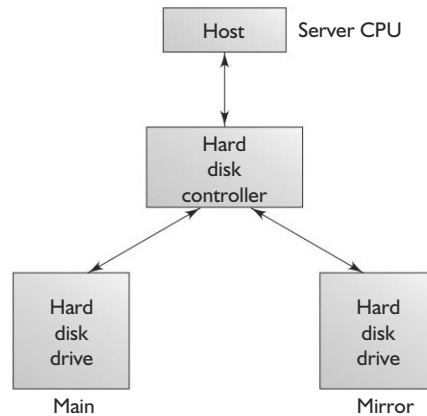
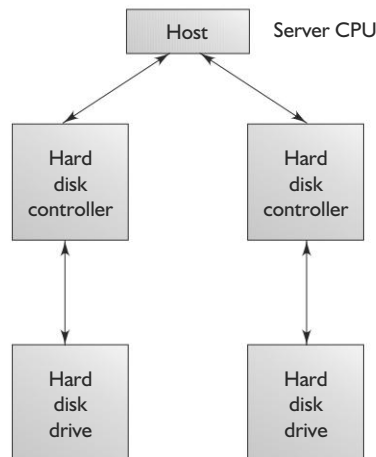
A server system is a special system whose software and hardware are designed for a specific use. Any computer executing applications or services for clients can technically be called a server. A variety of server systems are present. Some typical server types are:

1. Enterprise server
2. Network server
3. Web server
4. Database server
5. Applications server
6. Backup server
7. Cluster server
8. E-business server

16.8.1 Server System Technologies

A server system is supreme in following aspects:

1. Powerful computing with long word length, large main memory, high throughput and parallelism.
2. Higher clock speed.
3. Disk data reliability/Fault tolerance:

**Fig. 16.19** Disk mirroring**Fig. 16.20** Disk duplexing

- (a) Disk mirroring (Fig. 16.19)
- (b) Disk duplexing (Fig. 16.20)
- (c) Volume-drive spanning : spreading one logical volume into more than one physical drives.
- (d) Striping : writing data segments alternatively on different drives.
- (e) Redundant Array of Independent Disks (RAID): using two or more physical disk drives and using a combination of mirroring, duplexing, stripping etc.

4. Server fault tolerance:
- (a) Use of redundant power supplies with 'hot swap' feature
 - (b) Use of redundant fans with 'hot swap' feature
 - (c) Dual CPU system
 - (d) Dual server (server duplication; Fig. 16.21)
 - (e) Clustering (Fig. 16.22)



Fig. 16.21 Server duplication

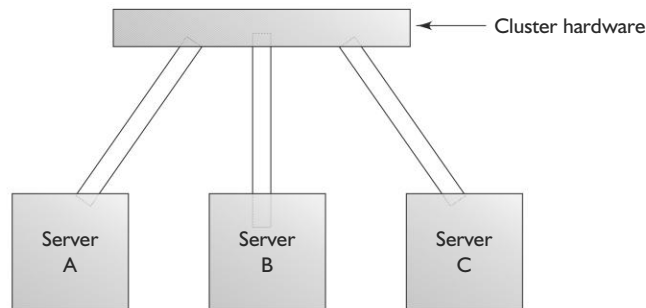


Fig. 16.22 Servers clustering

Some of these features are briefly discussed in the following sections with specific reference to IBM and SUN servers.

16.8.2 IBM Server Families

IBM offers a variety of server models:

1. Main frame servers (Z series, S/390 etc.)
2. Integrated applications servers
3. UNIX servers (p series)
4. Intel processors based servers
5. Clusters

The *p* series of the 660 Model 6M1 consist of a 64-bit Symmetric Multi-processing (SMP) Enterprise server which supports a range of 64-bit and 32-bit applications, simultaneously. It supports Hardware Multi-Threading (HMT) which involves sharing of a physical hardware processor by two logical processors. The overlapping of a memory access with processing is practiced for improving the overall system throughput.

The 64-bit RS64 IV processor supports a SMP configurations up to 8-way.

Clustering: Up to 32 Model 6M1 servers are supported in one clustered enterprise server system. The connection of two or more servers in a cluster has the following advantages:

1. Provides scalability
2. Sharing, replication, and redundancy within a cluster thereby enhancing the availability of resources
3. Single point management of the servers

The implementation aspects of Reliability, Availability and Serviceability (RAS) feature are:

1. Redundant power supplies
2. Hot-plug fans
3. Error recovery for the memory and caches by the ECC, which provides single bit error correction and double bit error detection.
4. Memory scrubbing: It involves continuous (hardware wise) readings of the memory in the background, for detecting correctable errors. It also reports to the service processor on exceeding its threshold.
5. Chipkill: The Chipkill memory protects the server from a single memory chip failure and multiple bit errors from any portion of the single memory chip.
6. Dynamic processor deallocation: It is followed in systems with more than two processors. It puts the failing processors off-line, without the operator intervention, and also reassigns the load to other processors. A defective processor is marked for deconfiguration at the next boot.
7. Persistent memory and processor deconfiguration: It is based on the failure history, marking of the processors and memory modules 'bad' to prevent their configuration on the subsequent boots.
8. I/O expansion recovery
9. PCI bus error recovery
10. Service processor: It is a diligent processor that monitors the system's health, detects the problems and provides advance warning. It is active using the standby power even when the system is powered-off. Its major functions are:
 - a. Automatic reboot under special situations.

- b. Surveillance of the operating system through a *heartbeat* mechanism and on sensing abnormal gap, carrying out the automatic reboot, thereby preventing the hang situation.
- c. Dial-out (call home) to report an error; dial-in even during the power-off allowing a remote session.

16.8.3 Sun Server Families

1. Sun Enterprise server family: A range of binary compatible servers that scale from 1 to 64 processors; designed for remote/branch offices and data centre environments;
2. Sun Fire family of servers based on UltraSPARC III processors; used as midframe.
3. Netra server family: meant for extreme environmental conditions.

16.8.3.1 Availability

It points to the time period when a particular resource (hardware or software) is accessible and usable. The entire system design, right from the core to the application software architecture, is accordingly developed for achieving a high level of availability. It is expressed as the percentage of total uptime over one full year. For example, a 99.9% availability indicates a down time of 8 hours per year. The following measures are taken to accomplish the goals:

1. Additional (redundant) power supplies to the base system
2. Additional (redundant) I/O controllers to the base system
3. Mirroring hard disks
4. Alternate pathing and dynamic configuration with the help of software
5. Adding an entire system as a hot spare (back up)

16.8.3.2 Dynamic Reconfiguration (DR) on Sun Enterprise servers

It is a unique feature that enables the system to replace and reconfigure the system hardware while it is operating. An engineer can add or replace the hardware modules such as CPU, memory and I/O interfaces with minimum interruption to a system's normal operation. When a system is running programs, in live condition, the system board can be made off-line, and removed, and another system board can be inserted, and made on-line. The advantages of DR are two fold: Elimination of the reboot after any hardware changes; Enhancement of the systems uptime (availability) since the system need not be halted or rebooted.

Hot plug PCI adapters: They support adding or removal of the I/O adapters (the Peripheral interfaces) when the system is powered on and operating.

SUMMARY

A multi-processor system consists of two or more processors. An interconnection network links the processors. The primary objective of a multi-processor system is to enhance the performance by means of parallel processing. It also offers fault tolerance. In a tightly coupled multi-processor, multiple processors share a common memory. The processors share information via the common memory. In a loosely coupled multi-processor system, there is no shared memory and each processor has its own private memory. Information is exchanged by different processors via the interconnection network by a message passing protocol. A symmetric multi-processor is a multi-processor system with identical processors of equal capabilities. All processors have equal access time for memory and I/O resources.

The loosely coupled multi-processor system has physically distributed memories. It is of two types: distributed shared memory multi-processor and cluster. In a distributed shared memory, the processors have a common, shared address space for all the memories. In a cluster, the processors do not share the address space. Clustering is interconnection of multiple independent computer systems functioning as a single system in a collaborative fashion. Two main objectives of forming a cluster are: load sharing and fault tolerance.

When more than one processor shares the main memory, copies of the main memory block are present in cache memories of all the processors. The cache coherence problem occurs if every processor is allowed to modify its cache without any precaution to maintain uniformity in all caches. Both the software and hardware methods are possible for the cache coherence problem.

Fault tolerance enables a computer to execute its operation correctly without getting affected by the hardware or software failures. To make a system fault-tolerant, special hardware and/or software modules are added to its design. Nowadays, simple fault-tolerant techniques are present even in the low cost computers.

Server systems are high performance systems for various specific applications. They offer high reliability.

Annexure 1



NUMBER SYSTEMS AND CONVERSIONS

A1.1 Introduction

A number system is a standard scheme for representing numeric values. Many types of number systems are practiced in the computer industry. Some number systems are used for easy communication/documentation of programs/data whereas others are used for internal representation of numbers in a computer. This annexure reviews different number systems and conversion of numbers from one number system to others.

A1.2 Common Number Systems

A number system with a base of n comprises of n different digits. A number can be formed using these digits in different positions of the number. Consider the following number:

$$d_m d_{m-1} \dots d_1 d_0 . f_1 f_2 \dots f_k$$

where $d_m, d_{m-1}, \dots, d_1, d_0, f_1, f_2, \dots$ and f_k are different digits. The value of the number is given by

$$(d_m \times n^m) + (d_{m-1} \times n^{m-1}) + \dots + (d_2 \times n^2) + (d_1 \times n^1) + (d_0 \times n^0) + (f_1 \times n^{-1}) + (f_2 \times n^{-2}) + \dots + (f_k \times n^{-k})$$

Table A1.1 lists common number systems and Table A1.2 shows the representation of 19 decimal numbers in these number systems.

TABLE A1.1 Number systems and bases

S.no.	Number system	Base	Purpose	Digits	Remarks used
1.	Decimal	10	Commonly used	0 to 9	Used in day-to-day life
2.	Binary	2	Internal representation in computers	0 and 1	Used inside computers
3.	Octal	8	Easy documentation of object program and data	0 to 7	Popular during minicomputer days; currently obsolete
4.	Hexadecimal	16	Easy documentation of object program and data	0 to 9 and A to F	Currently used widely

TABLE A1.2 Number systems and representations

Decimal number	Binary	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12

A1.3 Decimal Number System

We use decimal number system in day-to-day life. It is an easy to use system, with 10 different digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. The numeric value of the decimal number $d_m d_{m-1} \dots d_1 d_0 . f_1 f_2 \dots f_k$ is

$$(d_m \times 10^m) + (d_{m-1} \times 10^{m-1}) + \dots + (d_2 \times 10^2) + (d_1 \times 10^1) + (d_0 \times 10^0) + (f_1 \times 10^{-1}) + (f_2 \times 10^{-2}) + \dots + (f_k \times 10^{-k})$$

A1.4 Conversion of Decimal Numbers into Other Number Systems

For converting a number from the decimal number system into any other number system, separate it into integer part (left side of the decimal point) and fraction part (right side of the decimal point). For the integer part, the following procedure is followed:

1. Divide the integer part of the decimal number by the base of the target number system and note the remainder, R.
2. Divide the quotient (Q) of previous division by the base of the target number system and note the remainder.
3. Repeat step 2 till the quotient becomes 0.
4. Arrange the remainders in an order starting with the last division.
5. The number obtained in step 4 is the integer part of the desired number.

For the fraction part, the following procedure is followed:

1. Multiply the fraction part by the base of the system and note the integer part of the result.
2. Multiply the fraction part of the previous result by the base of the system and note the integer part of the result.
3. Repeat step 2 till the fraction part of the result is zero or till the required accuracy (number of digits after the decimal point) is achieved.
4. Arrange the integer parts in the order starting with first multiplication.
5. The number obtained in step 4 is the fraction part of the desired number.

Example A1.1

Convert the decimal number 93 into

- (a) binary number
- (b) octal number
- (c) hexadecimal number

(a) Conversion into a binary number:

$$93 \div 2, Q = 46, R = 1$$

$$46 \div 2, Q = 23, R = 0$$

$$23 \div 2, Q = 11, R = 1$$

$$11 \div 2, Q = 5, R = 1$$

$$5 \div 2, Q = 2, R = 1$$

$$2 \div 2, Q = 1, R = 0$$

$$1 \div 2, Q = 0, R = 1$$

Binary number = 1011101

(b) Conversion into an octal number:

$$93 \div 8, Q = 11, R = 5$$

$$11 \div 8, Q = 1, R = 3$$

$$1 \div 8, Q = 0, R = 1$$

Octal number = 135

(c) Conversion into a hexadecimal number:

$$93 \div 16, Q = 5, R = 13 = D$$

$$5 \div 16, Q = 0, R = 5$$

Hexadecimal number = 5D

Example A1.2

Convert 0.125 from decimal to binary

$$0.125 \times 2 = 0.250$$

$$0.25 \times 2 = 0.50$$

$$0.50 \times 2 = 1.00$$

Binary number = 0.001



A1.5 Conversion of Other Systems into Decimal Number System

A number in the non-decimal number system is converted into a decimal number by multiplying each digit in the number by its decimal value and adding all the products.

Example A1.3

Convert the hexadecimal number FA2E into a decimal number

$$\begin{aligned}\text{FA2E} &= (15 \times 16^3) + (10 \times 16^2) + (2 \times 16^1) + (14 \times 16^0) \\ &= (15 \times 4096) + (10 \times 256) + (2 \times 16) + (14 \times 1) \\ &= 61440 + 2560 + 32 + 14 \\ &= 64046 \\ \text{FA2E}_{16} &= 64046_{10}\end{aligned}$$

Example A1.4

Convert the octal number 765 into a decimal number

$$\begin{aligned}765 &= (7 \times 8^2) + (6 \times 8^1) + (5 \times 8^0) \\ &= (7 \times 64) + (6 \times 8) + (5 \times 1) \\ &= 448 + 48 + 5 \\ &= 501 \\ 765_8 &= 501_{10}\end{aligned}$$

Example A1.5

Convert the binary number 1011001 into a decimal number

$$\begin{aligned}1011001 &= (1 \times 2^6) + (0 \times 2^5) + (1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) \\ &= (1 \times 64) + (0 \times 32) + (1 \times 16) + (1 \times 8) + (0 \times 4) + (0 \times 2) + (1 \times 1) \\ &= 64 + 0 + 16 + 8 + 0 + 0 + 1 \\ &= 89 \\ 1011001_2 &= 89_{10}\end{aligned}$$

Example A1.6

Convert the binary number 101.01 into a decimal number

$$\begin{aligned}101.01 &= (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) + (0 \times 2^{-1}) + (1 \times 2^{-2}) \\ &= 4 + 0 + 1 + 0 + (0.25) = 5.25 \\ 101.01_2 &= 5.25_{10}\end{aligned}$$

Annexure 2



TYPES OF COMPUTER SOFTWARE



A2.1 Introduction

At the lowest level, software is a set of instructions in a machine language. At the highest level, software is a set of statements in a high-level language. Traditionally, based on its role, software is classified into system software and application software. As per recent trends, we identify three types of software: System software, Programming software and Application software. System software help the programmer manage the hardware resources in the computer system. For a given type of computer hardware, the system software may be unique and it can not be used in another type of computer system unless some changes are done. Programming software are tools to a programmer to write computer programs in different programming languages easily and rapidly. Application software helps the user perform a particular task. The objective of early period's application software was to simplify manual tasks. For example, word processing software replaced typing on paper, spreadsheets replaced physical ledgers, and database programs replaced hard copy-filing. The objective of present day application software is to advance the computer-based world, including media players, web browsers and electronic publishing tools.



A2.2 System Software Types

The system software relieves the application programmer from the burden of knowing the details of the particular computer being used and help the programmer manage the hardware resources in the system. The system software interfaces with hardware to provide the necessary services for application software. Examples are operating systems, device drivers,

servers, utilities, windowing systems, etc. The user deals with the operating system for any services he needs instead of accessing the hardware directly. Some popular operating systems are Windows XP, Linux, and Macintosh OS.

A2.3 Programming software

Programming softwares are tools to develop software using different programming languages. Examples are compilers, debuggers, interpreters, linkers, text editors, etc. An Integrated development environment (IDE) is a single application that manages all these tools, minimizing programmer's efforts.

A2.4 Application Software

Application software helps the user to perform a particular task. Depending on the design, an application program can manipulate text, numbers, graphics, or a combination of these items. Typical examples are word processors, spreadsheets, media players, database applications, industrial automation, business software, computer games, telecommunications, educational software, medical software, military software, image editing, simulation software and decision making software. The different programs cover a wide range from the home user to large organizations and institutions. In some types of embedded systems, the application software and the operating system software may be integrated, as in the case of software used to control a VCR, DVD player or microwave oven. Table A2.1 identifies the role of some popular types of Application Software and gives some examples for each type.

Application suite: An application suite consists of multiple applications bundled together. These applications generally have related functions, features and user interfaces, and can interact with each other. Examples of this are Microsoft Office, (which gives users Word, Excel, PowerPoint and Outlook) and Adobe Creative Suite (Photoshop, InDesign, Illustrator and Acrobat).

TABLE A2.1 Application software types and examples

Sl. no.	Application software type	Function/Role	Specific examples
1.	Word Processing Software	Enables creation and editing of documents	MS-Word, wordPerfect, WordPad and Notepad
2.	Database Software	Organizes the data into a database and performs database operations. Allows storing and retrieving of data. A database can be used to easily sort and organize records.	Oracle, MS Access
3.	Spreadsheet Software	A tool to perform calculations such as budgeting, forecasting, etc. It simulates paper worksheets.	MS Excel, Quattro Pro, Lotus 1-2-3, MS Works, AppleWorks
4.	Multimedia Software	Allows creation and playing of audio and video media	Audio converters, players, burners, video encoders and decoders, Real Player and Media Player
5.	Presentation Software	Displays information in the form of a slide show. Allows three functions: (a) editing of text (b) inclusion of graphics in the text and (c) executing the slide shows	Microsoft PowerPoint, HyperStudio, Flash, Director, HyperCard, Digital Chisel, SuperCard, Corel Envoy
6.	Enterprise Software	Performs customer relationship management or the financial processes. Deals with the needs of organization processes and data flow	Financial, Customer Relationship Management, and Supply Chain Management
7.	Departmental Software	Enterprise Software for smaller organizations or groups in a large organization	Travel Expense Management, and IT Helpdesk
8.	Enterprise infrastructure software	Has common capabilities to support Enterprise Software systems	Databases, Email servers, and Network and Security Management
9.	Information Worker Software	Handling projects within a department and individual needs of management of information	Documentation tools, resource management tools, time management and personal management systems

(Contd.)

Sl. no.	Application software type	Function/Role	Specific examples
10.	Educational Software	Helps in teaching and self-learning. Capable of conducting tests and tracking progress	Moodle
11.	Simulation Software	Simulates either physical or abstract systems useful for research, training or entertainment.	Flight simulators, scientific simulators
12.	Content Access Software	Used to access content without editing. Meets the needs in digital entertainment and publishing digital content	Media Players, Web Browsers, Help browsers, and Games
13.	Media development software	Generates print and electronic media in a commercial or educational setting	Graphic Art software, Desktop Publishing software, Multimedia Development software, HTML editors, Digital Animation editors, Digital Audio and Video composition
14.	Product engineering software	Developing hardware and software products	Computer aided design (CAD), computer aided engineering (CAE), computer language editing and compiling tools, Integrated Development Environments, and Application Programmer Interfaces
15.	Web based software	Executing software from the Internet	Online games, Virus protection software
16.	Desktop Publishing software	Enables making signs, banners, greeting cards, illustrative worksheets, newsletters, etc.	Adobe PageMaker, MS Word, MS Publisher, AppleWorks, MS Works, Quark Express
17.	Internet Browsers	Enables surfing the Web	Netscape Navigator, MS Internet Explorer, AOL Browser, mozilla firefox

(Contd.)

Sl. no.	Application software type	Function/Role	Specific examples
18.	Email software	Allows Sending and receiving email	MS Outlook Express, MS Outlook, Eudora
19.	Graphics Programs Works, MS Paint, Painter	Enables touch up of photographs and creating graphics (pixel-based)	Adobe Photoshop, Paint Shop Pro, AppleWorks, MS
20.	Graphics Programs (vector-based)	Creates graphics such as illustrations or cartoon drawings	Adobe Illustrator, Corel Draw, Apple Works, MS Works, MS Word
21.	Communications software	Enables two computers to communicate (using modems)	MS NetMeeting, AOL Instant Messenger, IRC, ICQ, CU-SeeMe

Annexure 3



ASCII CODE

ASCII (American Standard Code for Information Interchange) is a 7-bit code widely used in computers. Using ASCII code, 128 different characters covering the following are represented:

1. Upper case and lower case alphabets
2. Numerals
3. Punctuation marks
4. Special symbols
5. Control characters

ASCII, pronounced “ask-key”, is the popular code for Computer equipment. The standard ASCII table defines 128 character codes (from 0 to 127), of which, the first 32 are control codes (non-printable), and the remaining 96 character codes are representable characters. Tables A3.1 and A3.2 list both types of codes along with their hexadecimal equivalents and description. Table A3.3 lists the type definitions used in Table A3.1.

TABLE A3.1 Non-printable control codes of ASCII

Decimal	Hexadecimal	Symbol	Type	Description
0	0	NUL		Null
1	1	SOH	CC	Start of Heading
2	2	STX	CC	Start of Text
3	3	ETX	CC	End of Text
4	4	EOT	CC	End of Transmission
5	5	ENQ	CC	Enquiry
6	6	ACK	CC	Acknowledge
7	7	BEL		Bell (audible or attention signal)
8	8	BS	FE	Backspace

(Contd.)

Decimal	Hexadecimal	Symbol	Type	Description
9	9	TAB	FE	Horizontal Tabulation
10	A	LF	FE	Line Feed
11	B	VT	FE	Vertical Tabulation
12	C	FE	FE	Form Feed
13	D	CR	FE	Carriage Return
14	E	SO		Shift Out
15	F	SI		Shift In
16	10	DLE	CC	Data Link Escape
17	11	DC1		Device Control 1
18	12	DC2		Device Control 2
19	13	DC3		Device Control 3
20	14	DC4		Device Control 4
21	15	NAK	CC	Negative Acknowledge
22	16	SYN	CC	Synchronous Idle
23	17	ETB	CC	End of Transmission Block
24	18	CAN		Cancel
25	19	EM		End of Medium
26	1A	SUB		Substitute
27	1B	ESC		Escape
28	1C	FS	IS	File Separator
29	1D	GS	IS	Group Separator
30	1E	RS	IS	Record Separator
31	1F	US	IS	Unit Separator

TABLE A3.2 Printable codes of ASCII

Decimal	Hexadecimal	Symbol	Description
32	20		Space
33	21	!	Exclamation mark
34	22	"	Double quotes (or speech marks)
35	23	#	Number
36	24	\$	Dollar
37	25	%	Percentage
38	26	&	Ampersand
39	27	'	Single quote
40	28	(	Open parenthesis (or open bracket)
41	29	)	Close parenthesis (or close bracket)
42	2A	*	Asterisk
43	2B	+	Plus
44	2C	,	Comma
45	2D	-	Hyphen
46	2E	.	Period, dot or full stop
47	2F	/	Slash or divide

(Contd.)

Decimal	Hexadecimal	Symbol	Description
48	30	0	Zero
49	31	1	One
50	32	2	Two
51	33	3	Three
52	34	4	Four
53	35	5	Five
54	36	6	Six
55	37	7	Seven
56	38	8	Eight
57	39	9	Nine
58	3A	:	Colon
59	3B	;	Semicolon
60	3C	<	Less than (or open angled bracket)
61	3D	=	Equals
62	3E	>	Greater than (or close angled bracket)
63	3F	?	Question mark
64	40	@	At symbol
65	41	A	Uppercase A
66	42	B	Uppercase B
67	43	C	Uppercase C
68	44	D	Uppercase D
69	45	E	Uppercase E
70	46	F	Uppercase F
71	47	G	Uppercase G
72	48	H	Uppercase H
73	49	I	Uppercase I
74	4A	J	Uppercase J
75	4B	K	Uppercase K
76	4C	L	Uppercase L
77	4D	M	Uppercase M
78	4E	N	Uppercase N
79	4F	O	Uppercase O
80	50	P	Uppercase P
81	51	Q	Uppercase Q
82	52	R	Uppercase R
83	53	S	Uppercase S
84	54	T	Uppercase T
85	55	U	Uppercase U
86	56	V	Uppercase V
87	57	W	Uppercase W
88	58	X	Uppercase X
89	59	Y	Uppercase Y
90	5A	Z	Uppercase Z
91	5B	(	Opening bracket

(Contd.)

Decimal	Hexadecimal	Symbol	Description
92	5C	\	Backslash
93	5D	]	Closing bracket
94	5E	^	Caret - circumflex
95	5F	_	Underscore
96	60	`	Grave accent
97	61	a	Lowercase a
98	62	b	Lowercase b
99	63	c	Lowercase c
100	64	d	Lowercase d
101	65	e	Lowercase e
102	66	f	Lowercase f
103	67	g	Lowercase g
104	68	h	Lowercase h
105	69	i	Lowercase i
106	6A	j	Lowercase j
107	6B	k	Lowercase k
108	6C	l	Lowercase l
109	6D	m	Lowercase m
110	6E	n	Lowercase n
111	6F	o	Lowercase o
112	70	p	Lowercase p
113	71	q	Lowercase q
114	72	r	Lowercase r
115	73	s	Lowercase s
116	74	t	Lowercase t
117	75	u	Lowercase u
118	76	v	Lowercase v
119	77	w	Lowercase w
120	78	x	Lowercase x
121	79	y	Lowercase y
122	7A	z	Lowercase z
123	7B	{	Opening brace
124	7C		Vertical bar
125	7D	}	Closing brace
126	7E	~	Equivalency sign - tilde
127	7F		Delete

TABLE A3.3 Type definition

Type	Description
CC	Communication Control
FE	Format Effector
IS	Information Separator

Table A3.4 gives a condensed table for the 7-bit ASCII code. The msd corresponds to the most significant 3 bits of the ASCII code and the lsd corresponds to the least significant 4 bits of the ASCII code. Each item in the table is identified by the combination of column heading followed by the row heading. For example, the ASCII code for the character 'A' is 41. The 7-bit pattern for this character is 0100001.

TABLE A3.4 ASCII code condensed table

MSD LSD	0	1	2	3	4	5	6	7
0	NUL	DLE	SPACE	0	@	P	'	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	(	k	{
C	FF	FS	,	<	L	\	l	
D	CR	GS	-	=	M	)	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	-	o	DEL

The 7-bit ASCII code is represented in a byte with the MSB (D7) as 0. It is also possible to use bit D7 as the parity bit for the remaining seven bits.

In PCs, an 8-bit extended ASCII version representing 256 different characters is used. The additional 128 combinations have bit D7 as 1. These provide for special symbols, foreign language sets, block graphics characters, Greek letters, etc. Though the keyboard does not directly support these characters, the BIOS supports them as a combination of ALT key and numeric keys. For example, the Greek letter μ (mu) is typed by holding down ALT key and typing the numerals 230. This is because the extended ASCII code for μ is decimal 230 or Hexa E6.

The Extended ASCII Character Set also consists of 128 decimal numbers and ranges from 128 through 255 representing additional special, mathematical, graphic, and foreign characters. There are several different variations of the 8-bit ASCII table. The table A3.5 is as per ISO 8859-1, also called ISO Latin-1. Codes 129-159 contain the Microsoft® Windows Latin-1 extended characters.

TABLE A3.5 Extended ASCII Character set - ISO Latin-1

Decimal	Hexadecimal	Symbol	Description
128	80	•	Euro sign
129	81		
130	82	,	Single low-9 quotation mark
131	83	ƒ	Latin small letter f with hook
132	84	„	Double low-9 quotation mark
133	85	...	Horizontal ellipsis
134	86	†	Dagger
135	87	‡	Double dagger
136	88	^	Modifier letter circumflex accent
137	89	‰	Per mille sign
138	8A	Š	Latin capital letter S with caron
139	8B	‹	Single left-pointing angle quotation
140	8C	Œ	Latin capital ligature OE
141	8D		
142	8E	Ž	Latin capital letter Z with caron
143	8F		
144	90		
145	91	‘	Left single quotation mark
146	92	’	Right single quotation mark
147	93	“	Left double quotation mark
148	94	”	Right double quotation mark
149	95	•	Bullet
150	96	—	En dash
151	97	—	Em dash
152	98	~	Small tilde
153	99	™	Trade mark sign
154	9A	š	Latin small letter S with caron
155	9B	›	Single right-pointing angle quotation mark
156	9C	œ	Latin small ligature oe
157	9D		
158	9E	ž	Latin small letter z with caron
159	9F	Ÿ	Latin capital letter Y with diaeresis
160	A0		Non-breaking space
161	A1	¡	Inverted exclamation mark
162	A2	¢	Cent sign
163	A3	£	Pound sign
164	A4	¤	Currency sign
165	A5	¥	Yen sign
166	A6	¦	Pipe, Broken vertical bar
167	A7	§	Section sign

(Contd.)

Decimal	Hexadecimal	Symbol	Description
168	A8	¨	Spacing diaeresis - umlaut
169	A9	©	Copyright sign
170	AA	ª	Feminine ordinal indicator
171	AB	«	Left double angle quotes
172	AC	¬	Not sign
173	AD		Soft hyphen
174	AE	®	Registered trade mark sign
175	AF	—	Spacing macron - overline
176	B0	°	Degree sign
177	B1	±	Plus-or-minus sign
178	B2	²	Superscript two - squared
179	B3	³	Superscript three - cubed
180	B4	´	Acute accent - spacing acute
181	B5	µ	Micro sign
182	B6	¶	Pilcrow sign - paragraph sign
183	B7	·	Middle dot - Georgian comma
184	B8	¸	Spacing cedilla
185	B9	¹	Superscript one
186	BA	º	Masculine ordinal indicator
187	BB	»	Right double angle quotes
188	BC	¼	Fraction one quarter
189	BD	½	Fraction one half
190	BE	¾	Fraction three quarters
191	BF	¿	Inverted question mark
192	C0	À	Latin capital letter A with grave
193	C1	Á	Latin capital letter A with acute
194	C2	Â	Latin capital letter A with circumflex
195	C3	Ã	Latin capital letter A with tilde
196	C4	Ä	Latin capital letter A with diaeresis
197	C5	Å	Latin capital letter A with ring above
198	C6	Æ	Latin capital letter AE
199	C7	Ç	Latin capital letter C with cedilla
200	C8	È	Latin capital letter E with grave
201	C9	É	Latin capital letter E with acute
202	CA	Ê	Latin capital letter E with circumflex
203	CB	Ë	Latin capital letter E with diaeresis
204	CC	Ì	Latin capital letter I with grave
205	CD	Í	Latin capital letter I with acute
206	CE	Î	Latin capital letter I with circumflex
207	CF	Ï	Latin capital letter I with diaeresis
208	D0	Ð	Latin capital letter ETH
209	D1	Ñ	Latin capital letter N with tilde
210	D2	Ò	Latin capital letter O with grave
211	D3	Ó	Latin capital letter O with acute

(Contd.)

Decimal	Hexadecimal	Symbol	Description
212	D4	Ô	Latin capital letter O with circumflex
213	D5	Õ	Latin capital letter O with tilde
214	D6	Ö	Latin capital letter O with diaeresis
215	D7	×	Multiplication sign
216	D8	Ø	Latin capital letter O with slash
217	D9	Û	Latin capital letter U with grave
218	DA	Ú	Latin capital letter U with acute
219	DB	Û	Latin capital letter U with circumflex
220	DC	Ü	Latin capital letter U with diaeresis
221	DD	Ý	Latin capital letter Y with acute
222	DE	Þ	Latin capital letter THORN
223	DF	ß	Latin small letter sharp s - ess-zed
224	E0	à	Latin small letter a with grave
225	E1	á	Latin small letter a with acute
226	E2	â	Latin small letter a with circumflex
227	E3	ã	Latin small letter a with tilde
228	E4	ä	Latin small letter a with diaeresis
229	E5	å	Latin small letter a with ring above
230	E6	æ	Latin small letter ae
231	E7	ç	Latin small letter c with cedilla
232	E8	è	Latin small letter e with grave
233	E9	é	Latin small letter e with acute
234	EA	ê	Latin small letter e with circumflex
235	EB	ë	Latin small letter e with diaeresis
236	EC	ì	Latin small letter i with grave
237	ED	í	Latin small letter i with acute
238	EE	î	Latin small letter i with circumflex
239	EF	ï	Latin small letter i with diaeresis
240	F0	ð	Latin small letter eth
241	F1	ñ	Latin small letter n with tilde
242	F2	ò	Latin small letter o with grave
243	F3	ó	Latin small letter o with acute
244	F4	ô	Latin small letter o with circumflex
245	F5	õ	Latin small letter o with tilde
246	F6	ö	Latin small letter o with diaeresis
247	F7	÷	Division sign
248	F8	ø	Latin small letter o with slash
249	F9	ù	Latin small letter u with grave
250	FA	ú	Latin small letter u with acute

(Contd.)

Decimal	Hexadecimal	Symbol	Description
251	FB	û	Latin small letter u with circumflex
252	FC	ü	Latin small letter u with diaeresis
253	FD	ý	Latin small letter y with acute
254	FE	þ	Latin small letter thorn
255	FF	ÿ	Latin small letter y with diaeresis

Annexure 4



HARDWARE COMPONENTS AND ICs

A4.1 Introduction

The microprocessor and other LSIs together cannot build a complete computer. In addition to these, we also need a good number of simple ICs of SSI and MSI. A discussion of these ICs is the primary objective of this annexure. As a preparation for the readers, a review of the relevant concepts of digital electronics and integrated circuits is presented. The personal computer (PC) is taken as example wherever required.

A4.2 Hardware Components: Discrete and Integrated

The different hardware components used in a PC are shown in Fig. A4.1. The integrated circuit (IC) chip is a single package consisting of more than one component of the same or different types, forming a circuit. In short, IC means a circuit fabricated on a piece of semiconductor material. There are a variety of IC chips, each one offering a different circuit function, such as AND, OR, INVERT, NAND, NOR, etc. ICs offering such basic functions are known as GATES or LOGIC GATES. A gate is defined as a logic circuit with one or more input lines but only one output line. A single IC may also contain multiple gates of the same type. Similarly, an IC may contain a complex circuit formed by gates.

Classification of ICs

ICs are classified into SSI, MSI, LSI and VLSI according to the relative number of basic gates necessary to build a circuit to achieve the same logic function (as in ICs). Table A4.1 defines the SSI, MSI and LSI types with sample functions achieved at these levels.

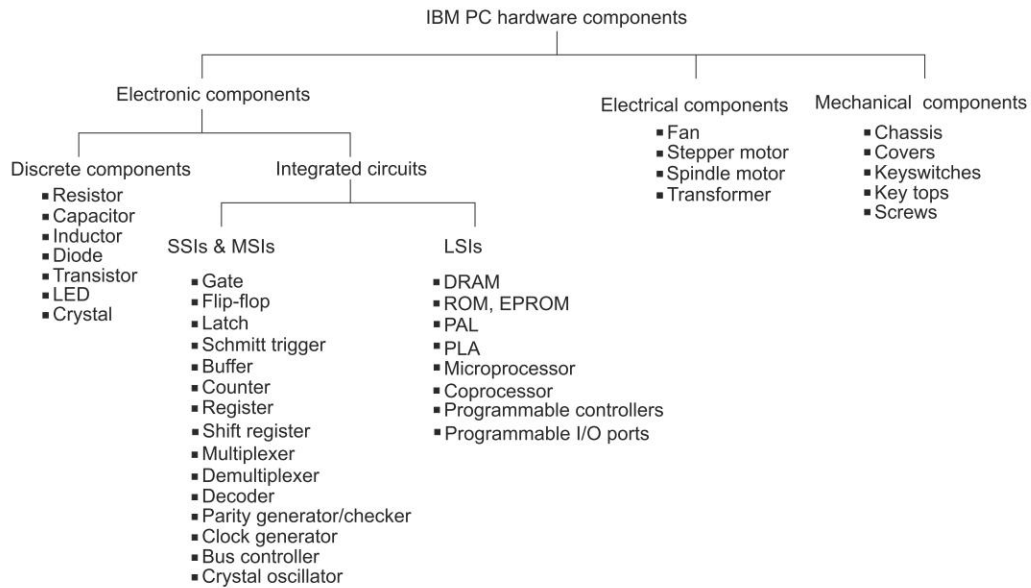


Fig. A4.1 Hardware Components in a PC

Table A4.1 Integration levels

Integration level	No. of Gates (IC Complexity)	Memory equivalent (bits)	Typical product
SSI (Small Scale Integration)	Less than 10	—	Quad 2 input NAND gate
MSI (Medium Scale Integration)	Between 10 and 100	Below 1k	J-K Flip Flop
LSI (Large Scale Integration)	Between 100 and 5000	1–16k	Programmable Controller
VLSI (Very Large Scale Integration)	Between 5000 and 50,000	> 16k	80286 Microprocessor
ULSI (Ultra Large Scale Integration)	Above 50,000	> 256k	256k SRAM



A4.3 Pulse Circuits and Waveforms

The main functions of pulse circuits are as follows:

1. Generation of pulse waveforms

2. Amplification of pulse waveforms
3. Shaping of pulse waveforms.
4. Storage of digital information
5. Switching
6. Gating

Figure A4.2 shows the characteristics of a typical pulse, which are defined below:

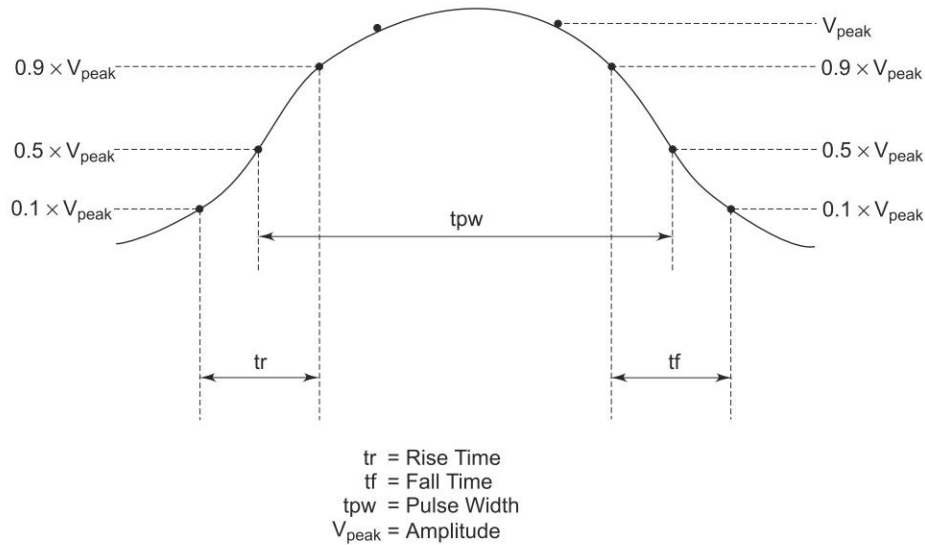


Fig. A4.2 Pulse characteristics

Rise Time: Time interval between 10% of the pulse amplitude (peak) and 90% of the pulse amplitude during rising edge.

Fall Time: Time interval between 90% of the pulse amplitude (peak) and 10% of the pulse amplitude during falling edge.

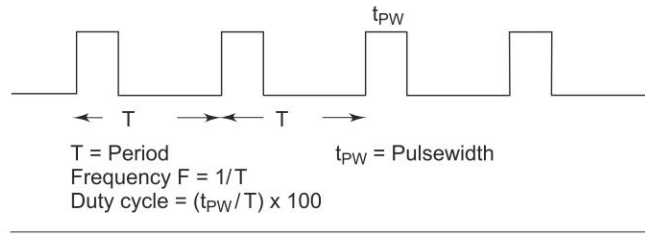
Pulse Width: The pulse width (tpw) indicates the duration of the pulse. It is measured as the time interval between the 50% points on the rising and falling edges.

Figure A4.3 illustrates the terms period, frequency, and duty cycle, which are defined below:

Frequency: The frequency (f) of a periodic waveform is the rate at which it repeats itself. It is specified as cycles per second (cps) or Hertz (Hz).

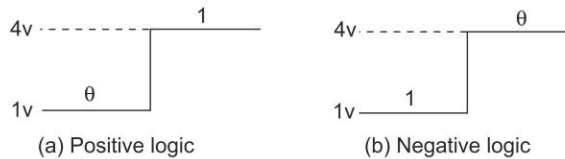
Period: The period (T) of a periodic waveform is the fixed interval at which it repeats. The relationship between the frequency and the period is $f = 1/T$.

Duty Cycle: The duty cycle is the ratio, in percentage, of the pulse width to the period, i.e., $\text{Duty Cycle} = (tpw/T) \times 100\%$.

**Fig. A4.3** Periodic waveform

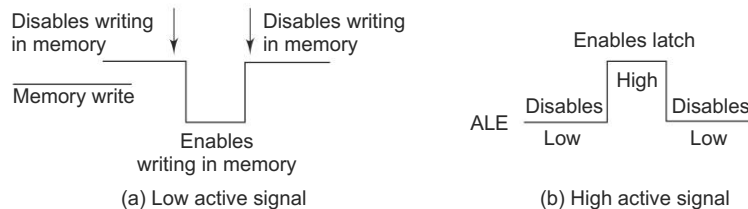
A4.4 Positive and Negative Logic

The terms positive logic and negative logic specify the way logic 1 and logic 0 are defined. In positive logic, the higher level is denoted as logic 1 and the lower level as logic 0. In negative logic, the higher level is denoted as logic 0 and the lower level as logic 1. Negative logic is very rarely followed. In a PC, it is used only in the RS232C communication interface. Figure A4.4 illustrates positive and negative logic.

**Fig. A4.4** Positive and negative Logic

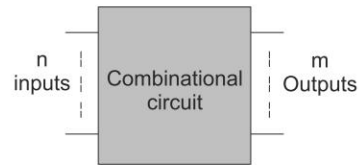
A4.5 High Active and Low Active Signals

When a high active signal is high, it carries some meaning and some action is done. When it is low, no action is done. When a low active signal is low, some action is done by it. When it is high, no action is done. Low active signals are identified by a minus sign before the signal name or a bar over it. Typical examples of high active and low active signals are illustrated in Fig. A4.5.

**Fig. A4.5** Low and high active signals

1. ALE (Address Latch Enable)

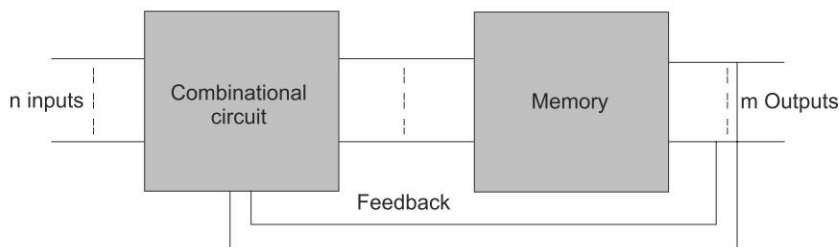
When this signal is 1, the address latch is enabled and address input enters into the latch. When the signal is 0, the address latch is disabled.

**Fig. A4.6** Combinational circuits**2. MEMW (Memory Write)**

When this signal is 0, memory takes the data present on the data bus and stores it. When this signal is 1, it has no effect.

**A4.6 Combinational and Sequential Circuits**

A combinational circuit always responds to a given input condition or combinations of it, in a constant way. The outputs at any time depend on the input only. Figure A4.6 illustrates a combinational circuit. A sequential circuit's output at any instant depends on the present inputs as well as the previous output before applying the current inputs. Figure A4.7 illustrates the sequential circuit.

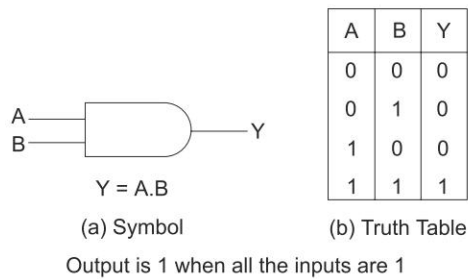
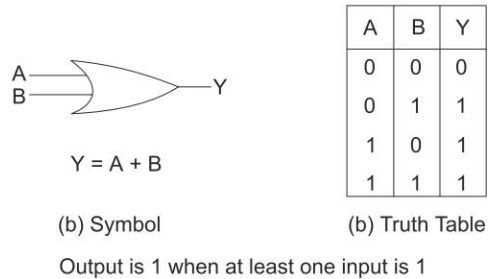
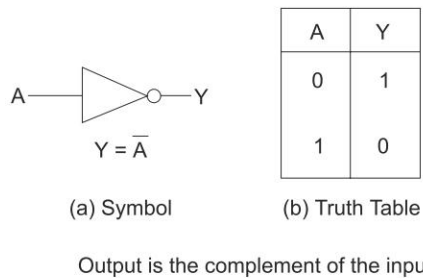
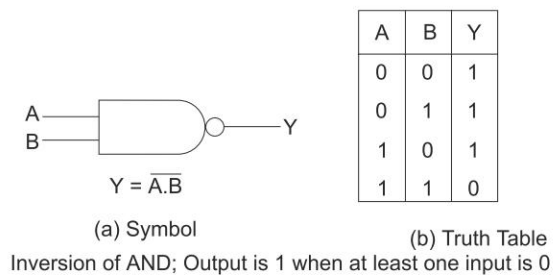
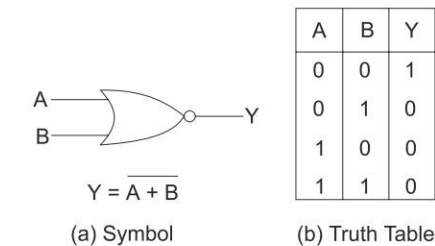
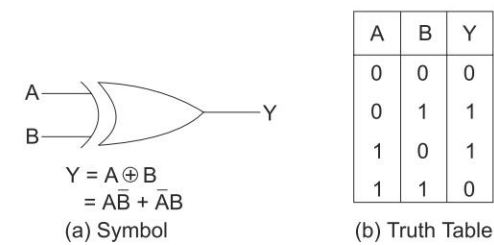
**Fig. A4.7** Sequential circuit

Gate, multiplexer, demultiplexer, decoder, encoder, parity generator, parity checker, comparator and adder are combinational circuits. Flip-flop, latch, counter and register are sequential circuits.

**A4.7 Simple Gates**

The symbols for simple gates AND, OR, NOT, NAND, NOR and EXCLUSIVE OR (XOR) are shown in Figs. A4.8, A4.9, A4.10, A4.11, A4.12 and A4.13. These figures also show the truth tables and function tables for these simple gates.

There are some more, one time heroes: AND-OR-INVERTER (AOI), EXPANDER, AOI with expandable input, etc. These are rarely used these days. Some of these are shown in Figs. A4.14 and A4.15.


Fig. A4.8 Two Input AND Gate

Fig. A4.9 Two Input OR Gate

Fig. A4.10 Inverter (NOT) Gate

Fig. A4.11 Two Input NAND Gate

Fig. A4.12 Two Input NOR Gate

Fig. A4.13 Exclusive OR (XOR) Gate

A4.8 Special Gates

There are some gates which do not perform any logic function. They are used merely for one of the following special functions:

1. Wave shaping
2. Increasing signal driving capacity
3. Removing noise on a signal (cleaning up)

4. Shifting signal levels
5. Interfacing between different IC families
6. Isolating an IC from its loads
7. Establishing bidirectional communication
8. Forming bus

These ICs are called by different names, such as BUFFER, DRIVER, LEVEL CONVERTER, LINE DRIVER, SCHMITT TRIGGER, etc., depending on the application.

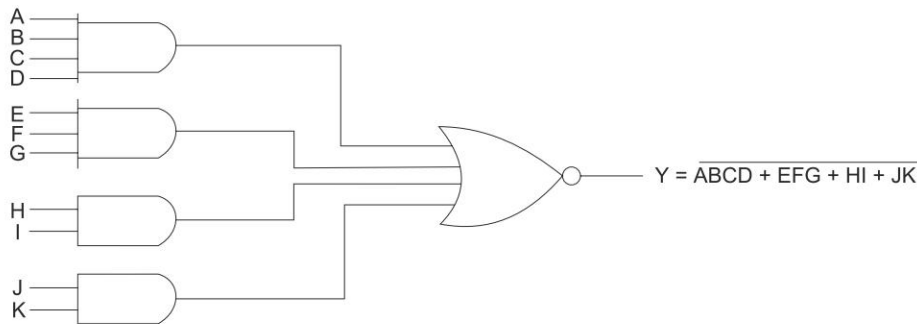


Fig. A4.14 4-3-2-2 Input AOI

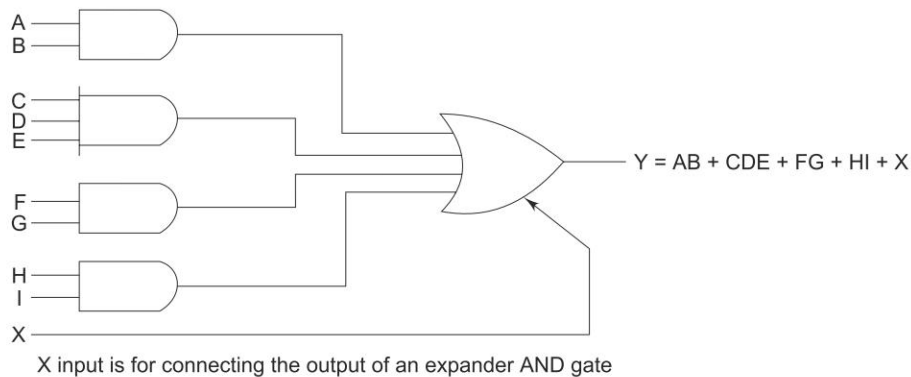


Fig. A4.15 AND-OR with expander (Expandable 4 Wide AND-OR)



A4.9 IC Families

Several types of ICs have been developed over the years. Popular IC families are listed as follows:

1. Resistor-transistor logic (RTL)
2. Diode-transistor logic (DTL)
3. Transistor-transistor logic (TTL)

4. Emitter-coupled logic (ECL)
5. High-threshold logic (HTL)
6. Metal oxide semiconductor logic (MOS)
7. Complementary metal oxide semiconductor logic (CMOS)
8. Integrated injection logic (IIL)

Different IC families differ in their characteristics, as listed below.

1. Speed
2. Power dissipation
3. Cost
4. Operating temperature range
5. Noise margin (Noise immunity)
6. Packaging density
7. Loading and driving limits
8. Ease of interfacing
9. Operating supply voltage
10. Ease of handling.

TTL logic and ECL logic have different circuit types but are both based on a common principle known as bipolar technology. The MOS logic is based on an entirely different principle known as unipolar technology. Table A4.2 compares TTL and MOS families. The ECL is not used in PCs. It is used only in large systems requiring very high speed operation.

Summarising, the advantages of TTL ICs are

1. High speed operation
2. Good driving/interfacing
3. Immunity to static electricity
4. Single supply voltage (+ 5V)

The main disadvantages of TTL ICs are

1. High power consumption
2. Large size for given function

The main advantages of MOS ICs are

1. Low power consumption
2. Small size
3. Wide power supply range
4. Good noise margin
5. Incorporation of resistors, capacitors and diodes inside the IC.

The disadvantages of MOS ICs are

1. Slow speed
2. Sensitivity to static electricity

Table A4.2 Comparison of TTL and MOS logics

Sl. no.	Parameter	TTL logic	MOS logic
1.	Propagation delay	As low as 5 ns for S series and 4 ns for ALS series gates. Hence circuits are faster	Very high; ranges from 15 ns to 300 ns for different MOS families. Circuits are slower
2.	Power dissipation	Higher; varies from 1 mW for L series to 22 mW for H series	Lesser; as low as 1 pW
3.	Power supply	Fixed; centered around + 5V	Operates over a wide range
4.	IC size or packaging density	Limited packaging density	High packaging density
5.	Noise Margin	400 mV only	As large as 45% of the power supply
6.	Fanout	Good; varies from 10 to 20	Poor; varies from 3 to 10
7.	Logic 1	Above 2V	Above 2/3 times the supply voltage
8.	Logic 0	Below 0.8V	Below 1/3 times the supply voltage
9.	Driving Capability	Good	Poor
10.	Immunity to static electricity	Strong; no special precautions are necessary	Weak; can be easily damaged by static electricity. Precautions during handling necessary
11.	Ease of interfacing with other families	Simple	Involved
12.	Operating temperature range	74 series: 0°C to 70°C; 54 Series: -55°C to + 125°C	-55°C to + 125°C
13.	Noise generation	High	Low
14.	Wired Logic	Yes; with open collector output gates	Nil
15.	Sensitivity to high voltages or short circuits	Very good; not damaged easily	Very poor; can be damaged easily

A4.9.1 TTL Subfamilies

Based on speed and power consumption, TTL subfamilies are named as follows:

1. High speed or high power TTL (H series)
2. Low power TTL (L series)
3. Schottky TTL (S series)
4. Low power schottky TTL (LS series)
5. Advanced low power schottky TTL (ALS series)

A comparison of the TTL subfamilies is presented in Table A4.3. The NAND gate is taken as the example.

Table A4.3 Comparison of TTL subfamilies: NAND

TTL Family	Standard	H	L	S	LS	AS	ALS
Fanout	10	10	10	10	20	—	20
Propagation delay (ns)	10	6	33	3	8	1.5	4
Power dissipation (mW)	10	22	1	19	2	8.5	1
Maximum Logic 0 Input current (mA)	1.6	2.0	0.18	2.0	0.4	—	0.2
Maximum Logic 1 Input current (μ A)	40	50	10	50	20	—	20
VOH min	2.4	2.4	2.4	2.7	2.7	—	2.5
VIH min	2.0	2.0	2.0	2.0	2.0	—	2.0
VIL max	0.8	0.8	0.8	0.8	0.8	—	0.8
VOL max	0.4	0.4	0.4	0.5	0.4	—	0.4

A4.10 TTL Characteristics

A4.10 Propagation Delay: t_{pd}

Any circuit takes sometime to react to the input. A change in inputs causes a change in the output, but after some delay. This delay is known as propagation delay. Figure A4.16 illustrates the propagation delay of an inverter gate. The average of t_{pLH} and t_{pHL} is taken as the average propagation delay of a gate.

A4.10.2 Power Dissipation

The power dissipation of a gate is the product of the dc supply voltage V_{cc} and the average supply current I_{cc} . The I_{cc} value is higher when a gate output is low (I_{ccL}) than when its output is high (I_{ccH}). The average I_{cc} is calculated from I_{ccL} and I_{ccH} assuming a 50% duty cycle operation, i.e., the low and high durations are equal.

A4.10.3 Fanout

The fanout of a TTL gate is related to the maximum current it can sink or supply. If the loading exceeds this, there is a risk of the driving gate getting damaged. Sometimes, when the loading marginally exceeds the limit, the driving gate may not fail totally, but the circuit will not work properly. A standard TTL gate can drive up to 10 standard TTL gates. The TTL gate has to supply $40\ \mu\text{A}$ to each load gate (standard TTL) when the output of the driving gate is high. When the output of the driving gate is low, each load gate delivers a maximum of $1.6\ \text{mA}$, which has to be sunk by the driving gate.

A4.10.4 Noise Margin

When several gates are interconnected to form a circuit, any noise superimposed over the input of a gate may create malfunctioning of the overall circuit, by pulling up or pulling down a signal. The noise margin is a measure of the noise tolerance level, or immunity to noise. In standard TTL circuits, a noise margin of 0.4V ($400\ \text{mV}$) is achieved by maintaining the following specifications:

1. Minimum voltage required at the input of a gate at high level = 2V , i.e., $V_{IH\ \text{min}} = 2\text{V}$
2. Minimum voltage supplied at the output of a gate at high level = 2.4V , i.e., $V_{OH\ \text{min}} = 2.4\text{V}$
3. Maximum voltage required at the input of a gate at low level is 0.8V , i.e., $V_{IL\ \text{max}} = 0.8\text{V}$
4. Maximum voltage supplied at the output of a gate at low level is 0.4V , i.e., $V_{OL\ \text{max}} = 0.4\text{V}$



A4.11 Open Collector TTL

Open collector TTL circuits were originally developed for bus applications. The advantage of the open collector circuit is a wired AND function, provided by connecting outputs

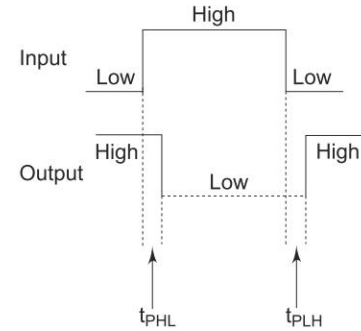


Fig. A4.16 Propagation delay

together. Figure A4.17 shows two 7407 gates (buffers with open collector output) whose outputs are tied together. The value of a pull up resistor depends on two factors:

1. Number of gate outputs connected together
2. Number of loads driven by gate outputs

The formula for calculating the value of the pull up resistor based on the above factors is given in IC data books/data sheets.

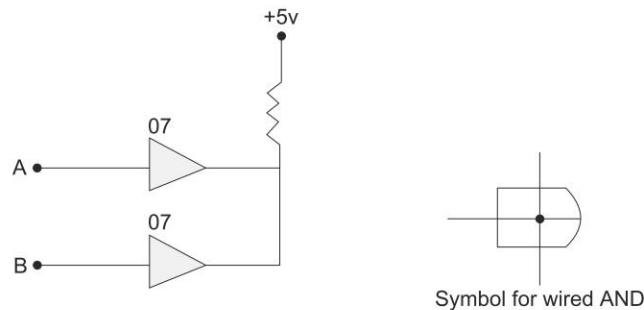


Fig. A4.17 Open-collector outputs on Bus

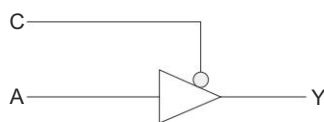


A4.12 Tri-State Gates

This logic has three output states:

1. Low-like normal TTL 0 state
2. High-like normal TTL 1 state
3. High impedance—neither high nor low.

The three-state gate (tri-state gate) has a separate control input. This input decides whether the output should be enabled (0 or 1) or tri-stated (disabled). In the floating output state, this gate appears to be an open circuit to the load. Figure A4.18 shows a three-state output buffer gate provided by a 74125 IC.



(a) Symbol

C	A	Y
0	0	0
0	1	1
1	0	Tri-state
1	1	Tri-state

(b) Truth table

Fig. A4.18 Tri-state buffer

Three-state output gates are useful in implementing a bus structure. On a common line, many three-state output gates are connected. At any time, only one of the gates is enabled and the enabled gate's output is 1 or 0. All other gates are disabled; the outputs of these gates are in a high impedance state. These gates are logically disconnected from the bus line and they do not draw any current from the load.

A4.13 MOS Families

MOS ICs are popular because of three main reasons: Low power dissipation, large functions in very little chip area and good noise immunity. Generally MOS logic ICs are LSIs, such as microprocessors, I/O controllers, memories, shift registers and ALUs. However, even SSIs and MSIs are available as MOS ICs to a limited extent. The popular MOS logic families are PMOS, NMOS, CMOS, DMOS, HMOS, VMOS and SOSMOS. The PMOS generally requires two supply voltages V_{gg} ($-27V$) and V_{dd} ($-13V$) in addition to ground (V_{ss}). The low threshold PMOS is an improved version of PMOS offering high speed and better TTL interfacing compatibility. This family requires three power supply voltages: V_{cc} ($+5V$), V_{dd} ($-5V$), and V_{gg} ($-12V$).

The NMOS has two versions. The older version requires three supplies: V_{cc} ($+5V$), V_{bb} ($-5V$), and V_{dd} ($+12V$). This version's output level is TTL compatible. The later version of NMOS requires only a single supply voltage of $+5V$. The VMOS, DMOS and HMOS are modifications of NMOS offering higher speed, comparable to ECL speed. The CMOS uses more chip area than the PMOS but offers higher speed and better TTL compatibility than PMOS. It requires a single supply voltage which can be chosen from a wide range (3 to 15V). The CMOS series has four subfamilies: 4000A, 4000B, Fairchild 4500 and National 74C00.

4000A is the original CMOS series. It offers medium speed like the PMOS, and its operating power supply voltage can be between 3V and $+15V$. The 4000B offers higher speed than the 4000A. Its supply voltage ranges from 3 to 18V. The 4000AU is similar to the 4000A but without an output buffer, and hence there is no extra propagation delay. Likewise, the 4000BU is similar to the 4000B without an output buffer, thereby reducing propagation delay. The 74C00 and 74HC00 are CMOS IC series with the same pinouts as 7400 series TTL. The 74C00 ICs are similar to the 4000B series, except that the pinouts of 74C00 are the same as corresponding ICs of the 7400 series TTL. The 74HC00 is a faster version of the 74C00 with pinout compatibility to 7400 series TTL.

A4.13.1 CMOS IC Characteristics

The main characteristics of CMOS ICs are summarised below.

Propagation Delay

CMOS ICs are slower than TTL ICs. Also CMOS ICs are sensitive to capacitive loading. The propagation delay decreases with increase in supply voltage. If the supply voltage is 15V, the propagation delay is less than 100 ns. If the supply voltage is 5V, the propagation delay is several hundred nanoseconds.

Supply Voltage

The supply voltage can be fixed anywhere between 3V and 15V.

Power Dissipation

The power dissipation is of the order of 10 μ W for a 5V power supply.

Noise Immunity

The noise immunity is approximately 45% of the supply voltage. For example, the noise margin is 6.75V for a 15V supply, 5.4V for a 12V supply, 2.25V for a 5V supply, and so on.

**A4.14 Interfacing Logic Families**

The output of an IC can be connected to the input of another provided both ICs are of the same family. If the ICs are of different families, appropriate interfacing has to be provided in order to take care of the driving and loading specifications of the output and input gates. This interface may require using a buffer gate in between the two gates, or extra pull up or series resistors in the path of the driving gate and the load.

A4.14.1 TTL Subfamilies

The input and output voltage levels of different subfamilies are compatible. There is no need for a special interface when one TTL subfamily gate is driving another TTL subfamily gate. But the fanouts of different TTL subfamilies differ. For example, the 74LS00 can drive only two standard TTL inputs.

A4.14.2 CMOS to TTL Interfacing

The fanout of the CMOS gate of 4000A is two standard TTL inputs of LS series. The 4000B and 74C00 series CMOS can drive one LS series TTL or two L series TTL loads. To exceed these limits, CMOS buffers like CD4049 or CD4050 should be used. These buffers provide the ability to source and sink enough output current.

A4.14.3 TTL to CMOS Interfacing

The minimum input voltage required at high level by a CMOS gate is about 3.6V, when operating at +5V. But the TTL gate output cannot usually provide this level. Hence a pull-up resistor is required on the output of the TTL, so that the output voltage at high state is

more than 3.6V. The value of the pull-up resistor depends on the number of CMOS inputs being driven by the TTL output. A 10 k Ω pull-up resistor is used in common circuits.

A4.15 Latch

Latch is a circuit used to latch or lock some signal information. It has two status conditions:

1. Latch open
2. Latch closed.

When the latch is open, the input signal will pass through the latch circuit. After latching, the circuit is insensitive to changes in the input signal, i.e., any more changes in the input signal have no impact on the circuit. The output of the latch at the instant of closing (latching) will continue after closing the latch also. Figure A4.19(a) shows the circuit for a $\bar{S} - \bar{R}$ latch using two NAND gates.

Table A4.4 $\bar{S} - \bar{R}$ Latch truth table

$\bar{S}$	$\bar{R}$	Q	$\bar{Q}$	Remarks	comments
0	0	—	—	—	Invalid inputs
0	1	1	0	SET	
1	0	0	1	RESET	
1	1	NC	NC	—	No Change

Table A4.4 gives the truth table of a $\bar{S} - \bar{R}$ latch. Either of the inputs going low latches the circuit. Subsequent change in that input does not affect the circuit. Both the inputs going low together leads to an invalid

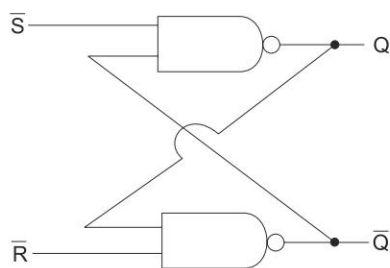


Fig. A4.19(a) $\bar{S} - \bar{R}$ Latch

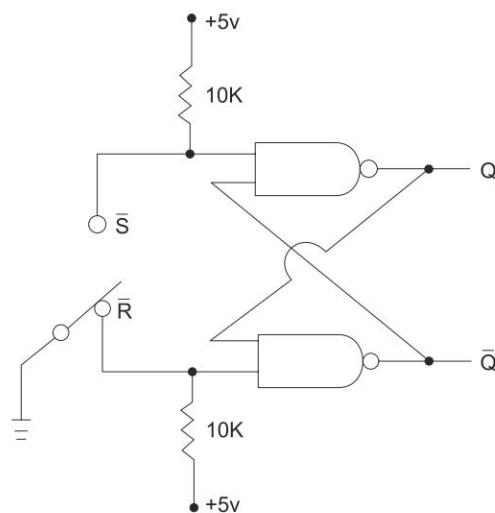


Fig. A4.19(b) $\bar{S} - \bar{R}$ Latch as a debouncer

condition, since after removing the inputs, the outputs return to the old state, i.e., the changes in the outputs are not permanent. When the $\bar{S}$ input goes low, the latch is set, i.e., Q becomes high and $\bar{Q}$ becomes low. If the $\bar{R}$ input goes low, the latch is reset, i.e., Q becomes low and $\bar{Q}$ becomes high. When both $\bar{S}$ and $\bar{R}$ are high, there is no change in the latch. The previous state of Q and $\bar{Q}$ continue.

The $\bar{S} - \bar{R}$ latch is used for eliminating the effect of bouncing of mechanical switch contacts. When a switch is operated, i.e., thrown from one position to another, the contact vibrates several times before finally sticking to the new position. The vibrations may continue for a few milliseconds. These vibrations will create chaos in circuits. Hence the vibrations should be killed. For this purpose, the $\bar{S} - \bar{R}$ latch can be used as a debouncing circuit, as shown in Fig. A4.19(b).

A4.15.1 Gated D Latch

The gated D latch is a modified $\bar{S} - \bar{R}$ latch with additional G input, as shown in Fig. A4.20. The truth table for the gated D latch is given in Table A4.5. When G is high, the output follows input. When G is low, the input does not affect the circuit. At the instant when G goes from high to low, the latch is closed. The output status of the latch at the low going edge of G is maintained subsequently, as long as the latch remains closed.

Table A4.5 Gated D latch truth table

D	G	Q	$\bar{Q}$	Remarks	Comments
0	0	NC	NC	Previous state remains	No change
0	1	0	1	Q follows D	
1	0			Previous state remains	
1	1	1	0	Q follows D	No change

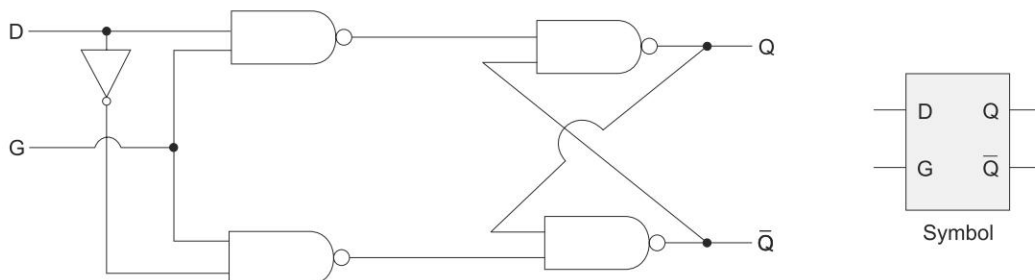


Fig. A4.20 Gated D latch

A4.15.2 Use of D Latch

One application of the D latch in the PC is for latching an address sent by the microprocessor like 8088. The microprocessor sends an address and ALE in T1. Then the microprocessor removes the address in T2. A gated D latch can be used as shown in Fig. A4.21 to latch the address.

A4.16 Flip-Flop

The flip-flop is a circuit providing one bit memory. A flip-flop is triggered by a clock signal due to which the synchronous data inputs enter the flip-flop. A flip-flop has two outputs, Q and $\overline{Q}$ which are complementary to each other. When the flip-flop is set, Q will be high and $\overline{Q}$ will be low. When the flip-flop is reset, Q will be low and $\overline{Q}$ will be high. When the flip-flop is triggered by the clock input, the flip-flop may remain unaffected or change over to the opposite state, depending on the synchronous data inputs.

The D flip-flop, J-K flip-flop, R-S flip-flop and the T flip-flop are some of the popular flip-flops. The D flip-flop and J-K flip-flop are used widely in PCs. The R-S flip-flop and T flip-flop have become obsolete.

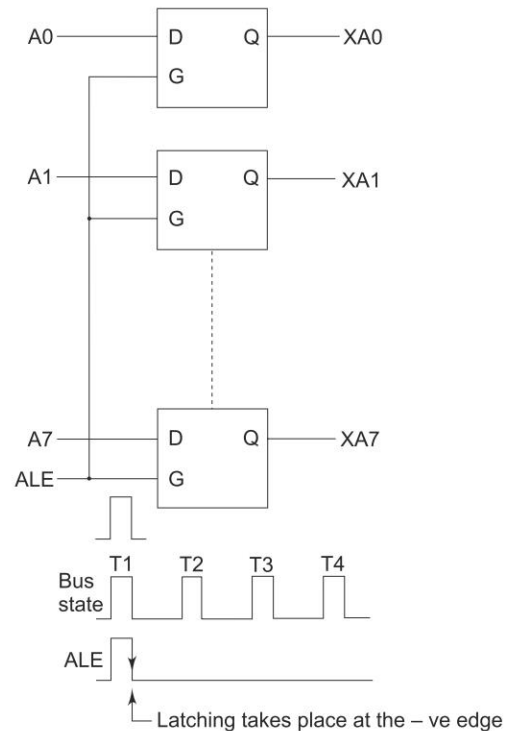
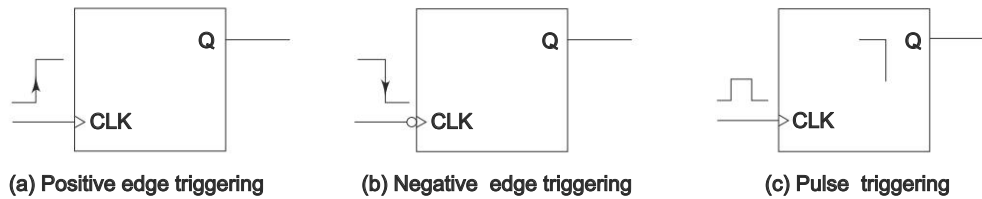


Fig. A4.21 Use of D latch

A4.16.1 Triggering Modes

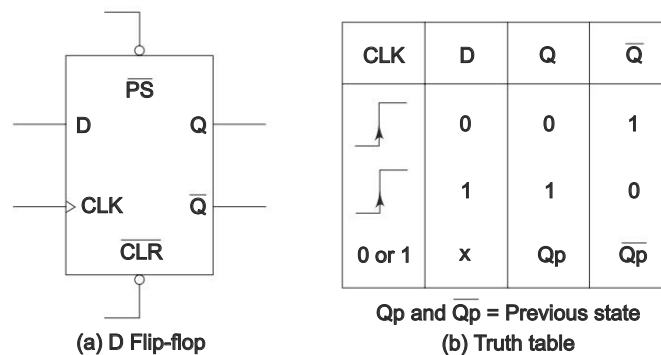
There are two modes of triggering a flip-flop: Edge triggering and pulse triggering.

In edge triggering, the positive going edge or negative going edge of the clock input triggers the flip-flop. In pulse triggering, a pulse at the clock input triggers the flip-flop. In a pulse triggered flip-flop, both edges are required to trigger the flip-flop. Edge triggered flip-flops are classified as positive edge triggered flip-flops and negative edge triggered flip-flops. Figure A4.22 shows the three types of triggering used in flip-flops.

**Fig. A4.22** Types of triggering

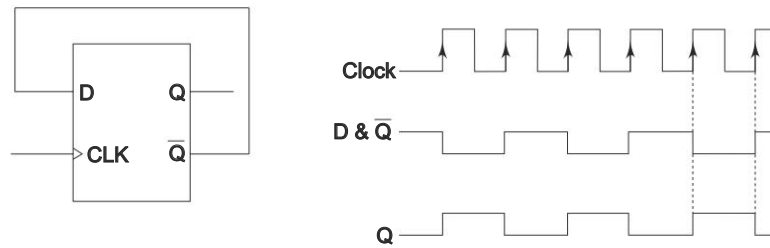
A4.16.2 D Flip-Flop

The D flip-flop is also called data flip-flop. When the flip-flop is triggered, the D (synchronous input) is transferred to the Q output. Figure A4.23 shows a positive edge triggered D flip-flop and truth table. When there is no positive edge at the clock input, the flip-flop remains unaffected.

**Fig. A4.23** D Flip-flop

This is shown in the truth table as Q_p and $\bar{Q}_p$ in the third row. Now the D can be 0 or 1 and this does not affect the flip-flop. This is indicated as X under D. The flip-flop is commonly used for the following purposes:

1. Temporary storage of signal information.
2. Transfer of data from one unit to another unit in a computer.
3. As flags to remember the occurrence of certain events, such as parity error, overflow, interrupt etc.
4. As a frequency divider, shown in Fig. A4.24



FF is initially assumed to be in reset state

Fig. A4.24 D Flip-flop as a frequency divider

Asynchronous Preset and Clear Inputs

The synchronous inputs affect the flip-flop during triggering only, whereas the asynchronous inputs can affect the flip-flop at any time. The $\overline{PS}$ denotes an asynchronous preset input. When $\overline{PS} = \text{low}$, the flip-flop is set. The $\overline{CLR}$ denotes an asynchronous clear input. When $\overline{CLR} = \text{low}$, the flip-flop is reset. When both $\overline{PS}$ and $\overline{CLR}$ inputs are simultaneously low, a temporary condition of both Q and $\bar{Q}$ becoming high happens. But once the $\overline{PS}$ and $\overline{CLR}$ are removed, the flip-flop returns to the original condition. The $\overline{PS}$ and $\overline{CLR}$ are known as asynchronous overriding inputs. They have higher priority over synchronous triggering.

A4.16.3 J-K Flip-Flop

The symbol and truth table for the positive edge triggered J-K flip-flop is shown in Fig. A4.25. If both J and K inputs are high during triggering, the flip-flop toggles. This feature is useful in counters wherein many J-K flip-flops are cascaded with common clock input. A single J-K flip-flop can also be used as a frequency divider by tying J and K together, as shown in Fig. A4.26.

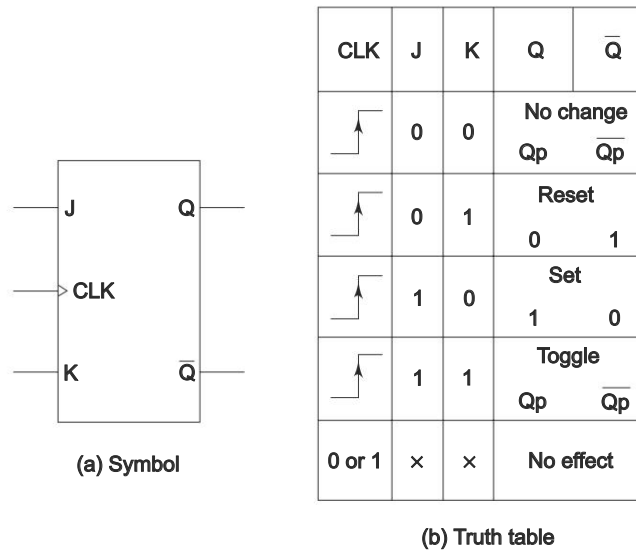


Fig. A4.25 J-K Flip-flop

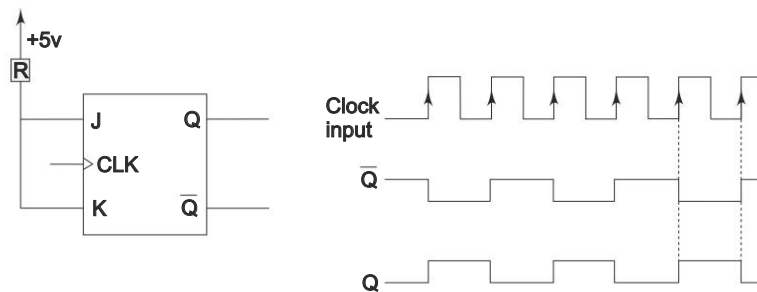


Fig. A4.26 J-K Flip-flop as a frequency divider

Difference between Latch and Flip-Flop

A flip-flop is triggered by a clock input. The data input (synchronous) to the flip-flop enters a flip-flop only when the flip-flop is triggered by the clock input. At all other times the data inputs have no effect on the flip-flop. In a latch, the data inputs enter as long as the latch is open, i.e., the gate input is high. When the gate input changes from high to low, the latch is closed. From here on, the inputs don't enter the latch.

Thus in a flip-flop, the inputs are ignored before triggering and after triggering. In a latch, the inputs are entertained before closing and ignored after it.

A4.17 Register

A register has multiple flip-flops with a common triggering clock input. For example, a four bit register implies that four flip-flops are connected in parallel with their clock inputs shorted. In addition, the $\overline{\text{CLR}}$ inputs of the individual flip-flops can be shorted together forming a common $\overline{\text{RESET}}$ input. Figure A4.27 shows a 4 bit D type register with a common clock and common $\overline{\text{RESET}}$ input.

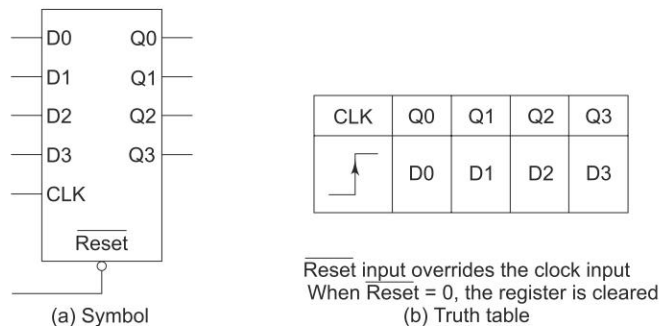


Fig. A4.27 4-bit register

A4.18 Shift Register

A shift register is a register whose contents can be shifted either left or right. There are shift registers with bidirectional shifting facility with control for left shift and right shift. The shifting of the contents by one bit (left or right) position is done when the shift register is triggered by the clock input. Figure A4.28 shows a 4 bit shift register with parallel loading facility.

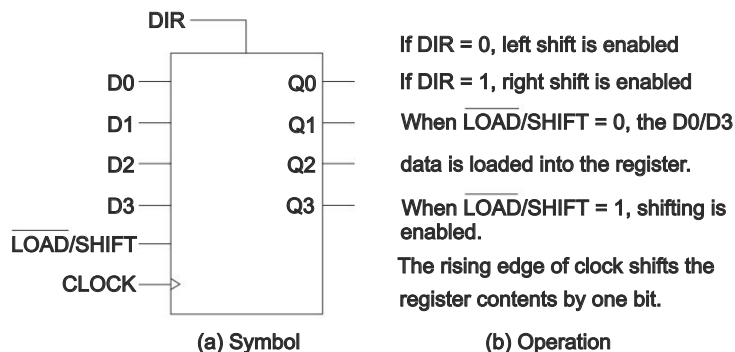


Fig. A4.28 4-bit shift register

A4.18.1 Shift Register Types

There are a variety of shift registers. The difference is in the nature of input and output. Figure 4.29 shows block diagrams for different types of shift registers, some of which are discussed below.

Serial-In Parallel-Out (SIPO)

When the $\overline{\text{RESET}}$ input is low, all the output bits become low; the shift register is cleared. When the first clock triggers, the data input sets or resets QA depending on whether IN is 1 or 0. During this, QB to QD outputs are unaffected. When the second clock arrives, the QA is shifted to QB and the data input is shifted to QA. This process continues for each clock. Thus after four clock inputs, a serial input bit stream is converted into a 4 bit parallel data.

Parallel-In Serial-Out (PISO)

When Shift/Load is low, the input data is entered into the shift register on getting a clock signal. When Shift/Load input is high and when the clock signal comes, the contents of the shift register are shifted out by one bit. Thus, during the first shift clock, the H bit comes to Q output. For the second shift clock, the G bit comes to Q output. In this way, an 8 bit parallel data can be converted into a serial bit stream output.



A4.19 Counters

The counter is a circuit with a set of flip-flops which counts the number of pulses given at the clock input. At any time, the number of pulses received (count value) is shown in the counter outputs, i.e., the output pattern in the flip-flops in different stages. Figure A4.30 shows a 4 bit binary counter. It counts from 0000 to 1111.

A4.19.1 Asynchronous Counter

In an asynchronous counter, the external clock input triggers only the first stage flip-flop. The second and further stages are triggered by the outputs of previous stages. All the flip-flops are not triggered simultaneously. A change in the output of a stage may trigger the subsequent stage. Thus triggering of the flip-flops in different stages takes place sequentially one after another. Due to this, the asynchronous counter is also termed ripple counter. Figure A4.31 shows a three stage asynchronous counter using J-K flip-flops.

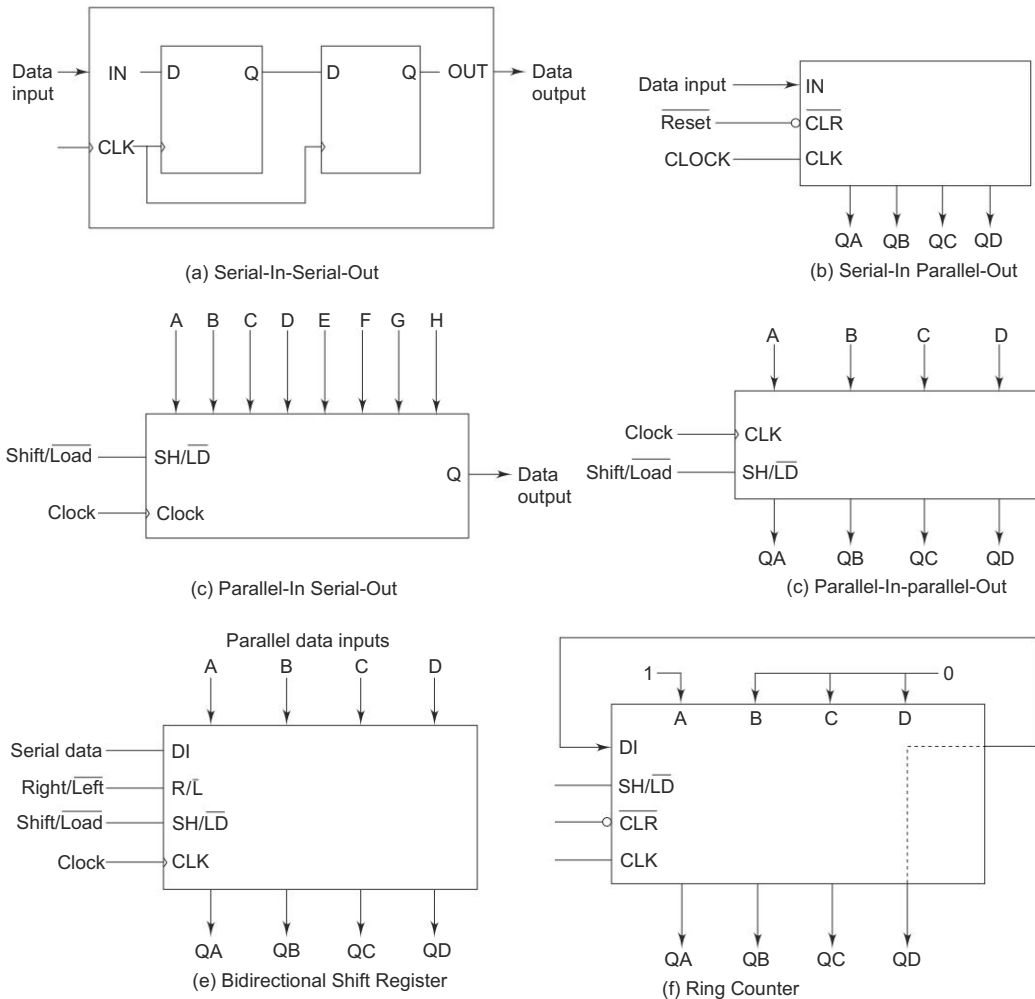
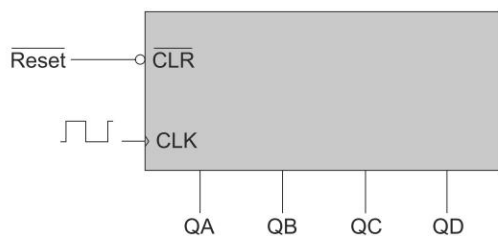


Fig. A4.29 shift register types



The QA, QB, QC & QD are the counter output bits. QA is the LSB and QD is the MSB. The RESET input, when LOW, clears the counter.

Fig. A4.30 4-bit binary counter

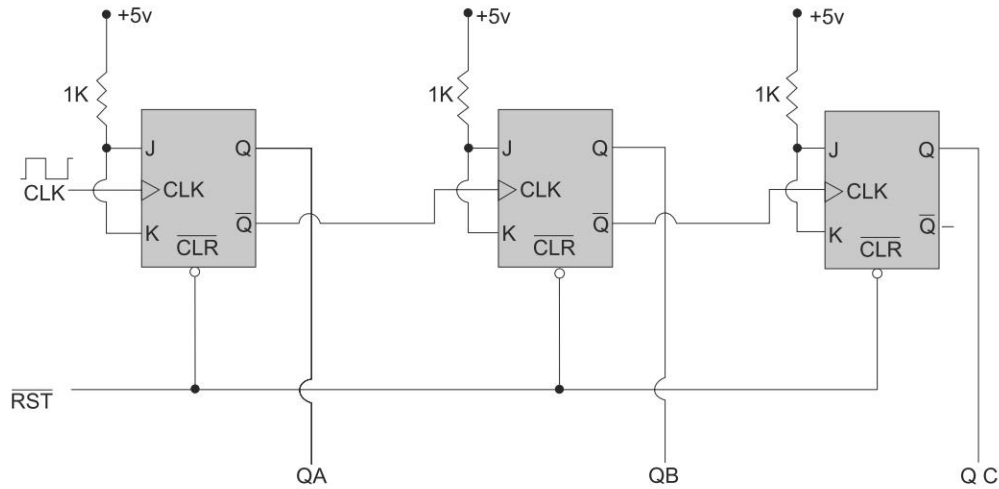
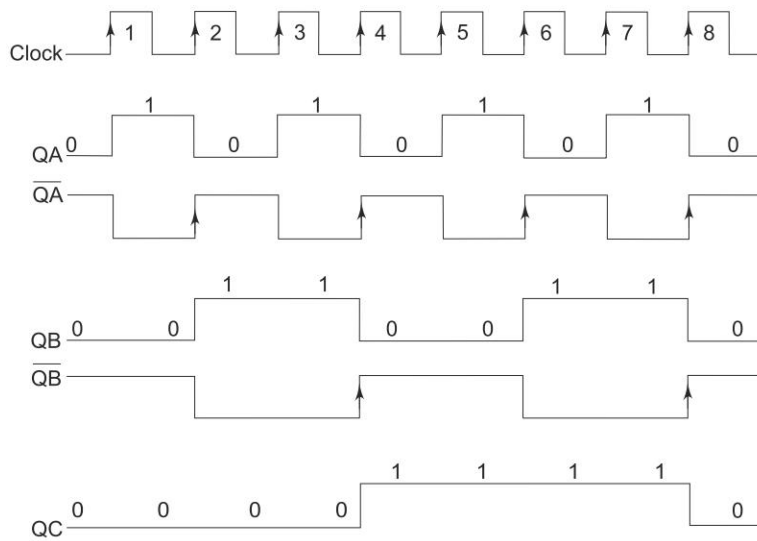


Fig. A4.31(a) Block diagram of 3-Stage asynchronous counter

A4.19.2 Synchronous Counter

In a synchronous counter, the external clock input triggers all the stages in the counter. Hence triggering of the flip-flops in different stages take place simultaneously. Figure A4.32 shows a three stage synchronous counter using J-K flip-flops.



(b) Timing Diagram

No. of Clock Pulses	Counter Output		
	QC	QB	QA
NIL	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8	0	0	0

(c) State Sequence

Fig. A4.31(b) 3-stage asynchronous counter

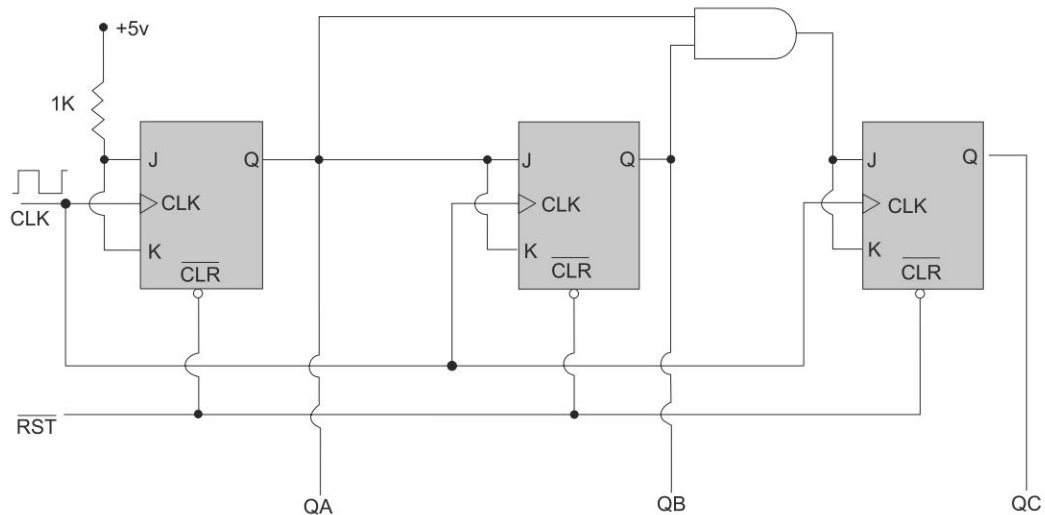
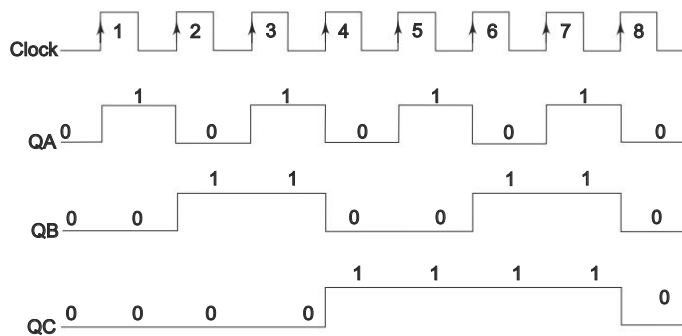


Fig. A4.32(a) Block diagram of 3-stage synchronous counter



(b) Timing Diagram

The first stage toggles for every triggering edge. The second stage toggles for the even number of clock pulses. The third stage toggles on the fourth and eighth clock pulse.

No. of Clock Pulses	Counter Output QC	QB	QA
NIL	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8	0	0	0

(c) State Sequence

Fig. A4.32(b) 3-stage synchronous counter



A4.20 Multiplexer

The multiplexer has multiple data input lines and one data output line. The function of the multiplexer is to select one of the input lines, and to connect the selected input to the output. Figure A4.33 shows the block diagram of multiplexer operation. If there are n bits in the

control inputs, a maximum of 2^n data inputs can be handled by the multiplexer. Table A4.6 illustrates the various control bit patterns and the corresponding input line selected by the multiplexer for an 8 to 1 multiplexer. In the 8 to 1 multiplexer, there are 8 input data lines and 3 control input lines.

Use of Multiplexer

The multiplexer can be used when there are many sources, all of which are connected to one destination. One example is memory addressing for the display memory (video buffer) in the MDA or CGA board. The display memory is a shared memory common to both the CPU (8088) and the CRT controller (6845). Figure A4.34 shows the video buffer multiplexing. For each address bit a 2 to 1 multiplexer is required. For all the multiplexers, the select/control inputs are connected together. If $A = 0$, CPU's address is selected. If $A = 1$, CRT controller's address is selected.

Another use of the multiplexer is to implement a parallel to serial converter required in serial communications.

In addition to data inputs and control lines, we have an extra input called strobe or enable.

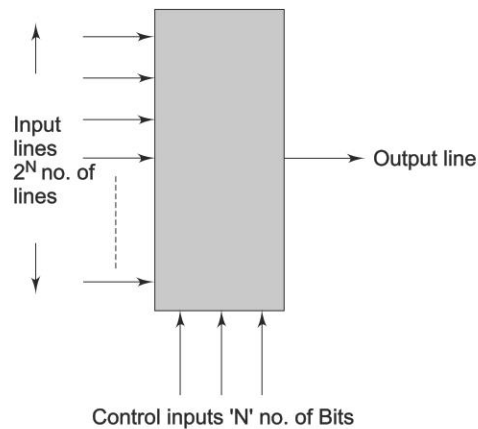
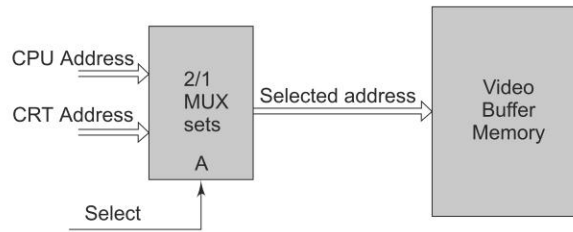


Fig. A4.33 Multiplexer block diagram

Table A4.6 Multiplexer Function Table

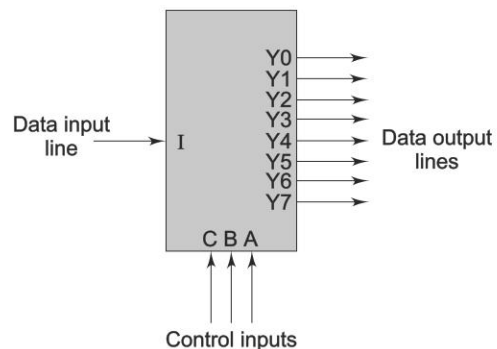
Select/Control input pattern			Selected input
C	B	A	
0	0	0	I0
0	0	1	I1
0	1	0	I2
0	1	1	I3
1	0	0	I4
1	0	1	I5
1	1	0	I6
1	1	1	I7

**Fig. A4.34** Video buffer multiplexing

This input is used to inhibit multiplexer operation. When the strobe input is high, the multiplexing action is disabled and the output is permanently low. Only when the strobe input is low, does the multiplexer select one of the inputs corresponding to the control bits pattern.

A4.21 Demultiplexer

The demultiplexer has one input line and multiple output lines. The function of the demultiplexer is to send the input on one of the output lines. The control bits pattern identify the output line on which the input has to be sent. Figure A4.35 shows the block diagram of the demultiplexer operation. If there are n bits in the control input lines, a maximum of 2^n output lines can be handled by the demultiplexer. Table A4.7 illustrates the various control bit patterns and the corresponding output line used by the demultiplexer for a 1 to 8 demultiplexer.

**Fig. A4.35** Demultiplexer block diagram

Use of Demultiplexer

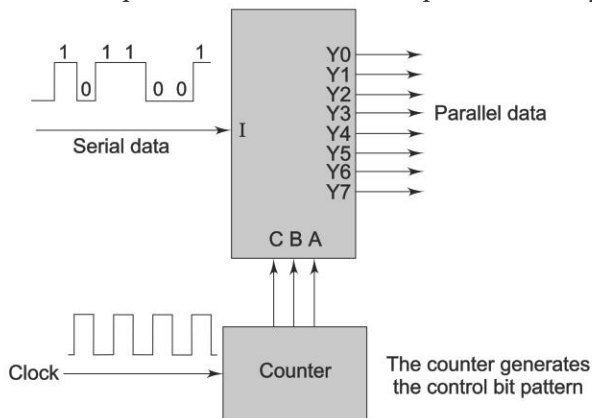
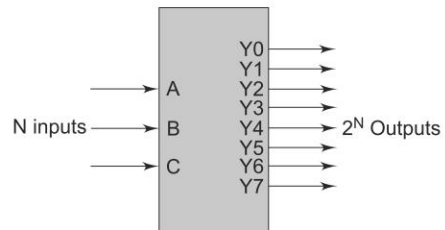
The demultiplexer can be used when there is one source but several destinations. The hardware applications of a demultiplexer include the following:

Table A4.7 Demultiplexer function table

Control input pattern			Selected output line
C	B	A	
0	0	0	Y0
0	0	1	Y1
0	1	0	Y2
0	1	1	Y3
1	0	0	Y4
1	0	1	Y5
1	1	0	Y6
1	1	1	Y7

1. Serial to parallel conversion of data from serial port.
2. Serial to parallel conversion of data in the floppy controller (or hard disk controller) during a read operation.

Figure A4.36 shows a simplified block diagram of a serial to parallel converter using a demultiplexer. A counter is used to generate the control bit pattern. The counter is a 3 bit counter, generating patterns 000 to 111 repeatedly. An increment of the counter from one control bit pattern to the next takes place for every clock pulse.

**Fig. A4.36** Serial-to-Parallel conversion**Fig. A4.37** Decoder block diagram

A4.22 Decoder

The decoder has n input lines and 2^n output lines. At any instant, only one of the output lines is active. The input bit pattern decides which output line is to be active at any instant. For each bit pattern on the input lines there is one output line. Thus the decoder decodes

the input bit pattern and explicitly identifies the pattern by making the corresponding output line active. Figure A4.37 shows the block diagram of the decoder operation. Table A4.8 gives the function table of a 3 to 8 decoder. Usually, the decoder outputs are active low; one of the output lines corresponding to the input pattern is low and all other output lines are high.

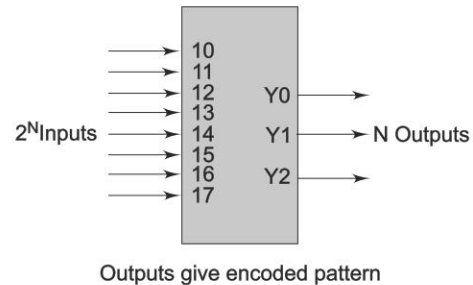


Fig. A4.38 Encoder block diagram

Table A4.8 Decoder function table

Input pattern			Active output
C	B	A	
0	0	0	Y0
0	0	1	Y1
0	1	0	Y2
0	1	1	Y3
1	0	0	Y4
1	0	1	Y5
1	1	0	Y6
1	1	1	Y7

Use of Decoder

The decoder is used in several places in the PC. Some of these uses are listed below:

1. Port address decoder
2. Command decoder
3. Device number decoder

These applications are discussed in detail in the subsequent chapters.



A4.23 Encoder

The encoder is the functional reverse of a decoder. The encoder has 2^n inputs and n outputs. At a given instant one of the inputs is active. The active input line is identified by a bit pattern on the output lines. Figure A4.38 shows the block diagram of an 8 input encoder and Table A4.9 gives the function table.

Use of Encoder

The encoder is used in applications where it is required to identify the active device signal out of multiple devices/signals. One popular example of the encoder concept is the keyboard encoder. The keyboard encoder has to identify which key has been pressed and generate a corresponding character code.

Table A4.9 Encoder function table

Input line active	Output pattern		
	Y2	Y1	Y0
I0	0	0	0
I1	0	0	1
I2	0	1	0
I3	0	1	1
I4	1	0	0
I5	1	0	1
I6	1	1	0
I7	1	1	1

A4.24 74S280 Parity Generator/Checker

This chip has a parity circuit which can be used both as a parity generator and as a parity checker. As a matter of fact, the circuit inside the chip always behaves as a parity generator.

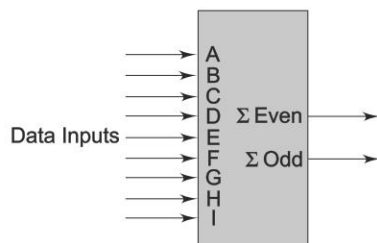


Fig. A4.39 74S280 block diagram

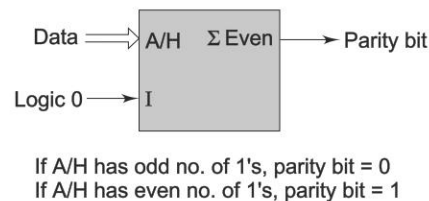


Fig. A4.40 Parity generator mode

By making appropriate input/output connections we achieve parity checking function also from this chip. Figure A4.39 shows the block diagram of the 74S280.

If the total number of 1's in the nine inputs is an odd number, the Σ ODD input pin becomes high, whereas the Σ EVEN output pin becomes low. On the other hand, if the total number of 1's in the nine inputs is an even number, then Σ EVEN output pin becomes high whereas the Σ ODD output pin becomes low.

In the PC, a single 74S280 chip is used, both as parity generator and as parity checker.

While writing data into DRAM, the 74S280 is used as a parity generator and while reading data from DRAM, the same chip is used as a parity checker. Figures A4.40 and A4.41 illustrate these two modes.

A4.24.1 Parity Generator Mode

The 8 bit data D0-D7, to be written into DRAM, is connected to the inputs A to H. The ninth input is kept low during writing. Since the ninth input is 0, it does not change the number of 1's in the other eight bits. Hence the chip is configured as parity generator for 8 bit data. The Σ EVEN output is used as the parity bit to be written in memory.

A4.24.2 Parity Checker Mode

The 8 bit data D0-D7 read from DRAM, is connected to the inputs A to H. The parity bit read from DRAM is connected to the ninth input, I. Now the chip behaves internally as a parity generator for nine inputs. If the nine inputs have odd parity, there is no parity error. If the nine inputs have even parity, there is a parity error. To achieve this, the Σ ODD output is used as parity error. If this output is high, there is no parity error; if low, there is a parity error.

The I input (pin no. 4) has to be supplied as logic 0 during parity generation and as parity bit during parity checking. This is achieved as shown in Fig. A4.42.

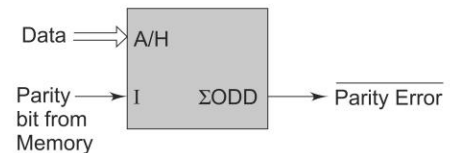


Fig. A4.41 Parity checker mode

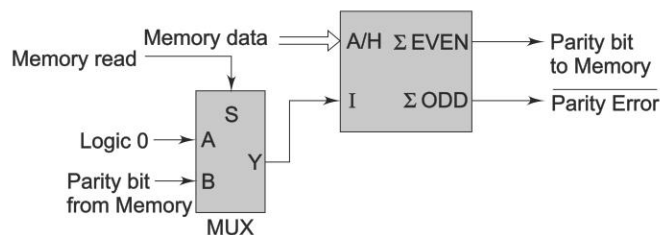


Fig. A4.42 74S280 as parity generator cum checker

A4.24.3 Error Correction Code (ECC)

In early days, ECC was used mainly for the hard disk. Presently ECC is used for the memory also. In simple ECC circuits, detection and correction of single bit errors take place without even the user knowing it. Complex memory controllers support detection and correction of multiple bit errors also.

Index

A

- Abacus 72
- Access time 18, 300, 414
- Accumulator architecture 64
- Accumulator based CPU 109, 110
- Adder 25
- Addition 204
- Address
 - bus 33, 502
 - marks 398
 - multiplexing 315
 - phase 560, 566, 567
 - translation 368
 - translation mechanism 373
- Addressing mode 102, 123, 647
- Advanced transfer cache 737
- Alpha 647
- Alpha family 738
- ALU 10
- Analog
 - computer 5
 - recording 388
- Antidependency 702
- Application program 9
- Arbiter 497, 559
- Arbitration phase 577, 579
- Architecture 55
- Arcuate scan 439
- Arithmetic and bit level parallelism 651
- Arithmetic
 - pipeline 658, 761
 - shift 97
- Array processing 759
- Array Processor 653, 769, 770
- Associative
 - mapping 349, 350

- memory 378, 379
- Asynchronous
 - communication 547
 - DRAM 322
 - information transfer 581
 - interface 21
 - transfer 508, 512, 514, 515
- Auto
 - configuration 560
 - shutdown 24
- Auxiliary memory 13, 302

B

- Backside bus 363
- Bad track 399
- Bandwidth 301
- Base register addressing 128
- Baud rate 547
- Benchmark
 - programs 41, 42
 - suite 41
- Biased exponent 146
- Big-endian assignment 116
- Binary arithmetic 139
- BIOS 8
- Bit cell 389
- Blade
 - enclosure 437
 - server 437
- Block Mode 496
- Blocked Mode 439
- Boot
 - record 36
 - sector 415
 - strap loader 35
- Booth's algorithm 210

- Booting 35
 - sequence 35
- Boundary scan 564
- Branch
 - delay slot 644
 - instruction results 677
 - prediction 704, 705, 736
 - slots 748
 - target buffer 699, 736
- Bridge 523, 557
- BTB 699
- Bubble memory 304
- Bubbles 670
- Built-In Self-Test 242
- Bundles 753
- Burst
 - access 565
 - mode 495, 557
- Bus 31, 500, 502
 - arbitration 496, 497, 515, 559
 - arbitration logic 490
 - cycle 34, 501, 502, 505, 515, 564
 - free phase 577, 579
 - master 496, 497, 564
 - mastering 559
 - states 506
 - structure 32, 500
- C**
- Cable
 - modem 635
- Cache
 - coherence 783
 - controller 343, 358
 - data memory 344
- DRAM 321
- flush 358
- flushing 366
- hit 345
- invalidation 367
- line 343, 344
- line fills 365
- line invalidations 366
- memory 44 303, 342, 343, 783
 - miss 345
 - replacement 366
 - snooping protocol 784
 - tag memory 344
 - write policy 355
- Calculators 72
- Capacitive Keyswitch 606
- Carry look-ahead adder 194, 197
- Cartridge 437
 - tape drive 442
- Cassettes 437
- CD-R 449
- CD-Recordable 449
- CD-Rewritable 450
- CD-ROM 447
- CD-RW 450
- CDC 6600 656, 657, 708
- Centralized bus arbitration 497
- Centronics interface 535, 537
- Character printer 616
- Chopping 229
- Circular shift 96
- CISC 47, 105, 106, 641, 648
- CISC architecture 641
- CISC processor 645, 648
- Clock 26
- Clocking 250
- Closed-loop
 - positioning system 409
 - system 410
- Clusters 782, 792
- Colour monitor 610
- Combinational
 - ALU 157
 - circuits 250
- Command
 - block 426
 - types 465
- Commit
 - stage 705
 - unit 700
- Compact Disc 447
- Compiler 7, 48, 746

- Compiler Efficiency 48
- Composite Video 611
- Composite video interface 609
- Computer
 - evolution 63
 - generations 75
 - organization 55
 - performance 39
 - structure 50
- Computing 3
- Condition codes 684
- Conditional
 - branch instruction 676, 678, 680, 682
 - branching 77
- Configuration 560
 - cycle 569
- Content address memory 369
- Content addressable memory 378
- Control
 - bus 33
 - flags 103
 - hazard 668, 676
 - point 27
 - signal 528, 536
 - transfer instructions 104
 - unit 10, 98, 100, 267
 - unit ddesign 267
- Coprocessor 170
- CPU 12
- Cray-1 767, 768
- CRC character 443
- Crossbar
 - switch 780
- CRT
 - display 608
 - interface 611
 - monitor 608
- Cycle
 - stealing 489
 - time 18, 300
- Cylinder 406

D

- Daisy
 - chain 497, 498
 - wheel printer 617

- Data
 - bus 32, 504
 - communication 546
 - dependency 671, 709
 - hazard 668, 671, 675
 - phase 560 566, 568
 - striping 432
 - transfer interrupt 468
- Dataflow Analysis 733
- Datapath 95, 98, 139, 161
- DCE 549
- DDR SDRAM 325
- Decimal arithmetic 109, 139
- Delay slot 680, 681, 748
- Delayed branch 680, 748
- Demand mode 496
- Depth 662
- Device
 - controller 14, 464
- Digital
 - camera 630
 - computer 5
 - recording 388
 - versatile disc 450
- Direct
 - addressing 125
 - I/O 462
 - mapping 346, 347
- Directory protocol 785
- Disk
 - duplexing 432
 - mirroring 432
- Distributed
 - bus arbitration 497
 - memory multi-processor 776
- Division 216
- DMA 481
 - channels 494, 495
 - controller 480, 484, 486, 492
 - mode 30, 484
 - parameters 485, 486, 492
 - sequence 487
- Dot matrix printer 618
- Double error 244

Draft quality printer 616
 DRAM 313
 DTE 549
 Dual address cycles 565
 DVD 450
 Dynamic
 branch prediction 699
 memory 305
 pipelining 700
 reconfiguration 796
 scheduling 700, 707, 708, 745

E

EAROM 306
 Edge triggered 250
 EDVAC 77
 EIDE 428, 429
 EISA 557
 Electronic computers 71
 Encoding 603
 End of I/O interrupt 468
 Endpoint 589
 Enhanced IDE 428
 ENIAC 76
 Enumeration 589
 EPIC 749
 EPIC architecture 745, 752
 EPROM 306
 Error correction 432
 Exception handling 684
 Exceptions 685
 Excess-127 code 146
 Execution
 time 39, 40
 unit 659
 Exponent 143
 overflow 232
 underflow 233
 External
 bus 34
 interrupts 467

F

False wait state 490
 Fast page mode 321

Fault tolerance 782, 786
 Fault-tolerant 788
 Fax-Msodem 628
 FDD specifications 403
 Fetch unit 658
 Fifth Generation Computers 84
 Firewire 555
 First Generation Computers 75, 76
 First-In First-Out (FIFO) algorithm 354
 Fixed-point arithmetic 140, 147
 Fixed-point number 140
 Flash memory 306
 Flat panel displays 633
 Floating-point
 addition 226
 arithmetic 109, 225
 division 233
 multiplication 230
 number 143
 pipeline 698
 subtraction 230
 Floppy
 disk 393
 controller 401
 drive 400
 format 397
 diskette 393
 Flynn's classification 651
 FM 389
 Formatting 397
 Fourth Generation Computers 83
 Fraction 140
 Frequency modulation 388
 Front side bus 363
 Full-adder 192
 Functional redundancy check 790

G

Gather operation 764
 Genuine wait state 490
 Golf-ball printer 618
 Graphic tablet 632
 Graphics accelerator 633, 634

Groups 753
Guard Digits 229

H

Half-adder 190
HALT 165
Halt state 24
Handshake
 packet 591
 protocol 513, 535
Handshaking 512
Hard disk
 controller 427
 drive 392, 404, 411, 413
 format 416
Hard sectoring 396
Hardware 6, 7
 faults 788
 line feed 539
 RAID 436
Hardwired control unit 67, 268
HDC commands 424
HDD interface 417
Head actuator 404
Head crash 409
Head disk assembly 413
Helical scan 439
History 699
History bit 700
Hit rate 345, 358
Hot
 pluggability 554
 spares 436
 swapping 432, 436, 585
HSYNC 608
Hubs 587, 589
Hybrid RAID 434
Hypercube network 781

I

I/O
 bus 500, 518, 519, 525
 controller 14, 455
 driver 16, 455, 459

 drivers 8, 458
 mapped I/O 327, 462
 port 30, 460
 read 505
 techniques 29, 31, 69, 480
 write 505
IA64 architecture 753
IA64 processor 753
IAS computer 77
IBM 80
IBM server families 794
ID field 397, 399
IDE 428
IEEE 1284 542, 543
ILLIAC IV 770
ILP 707
Immediate addressing 125
Impact printer 616
IN instruction 30, 460
In-order retirement 709
Incrementer 151
Index
 addressing 129
 hole 394
 sensor 394
Indirect addressing 126
Information transfer phase 581
Initiator 429, 560, 564, 566, 573
Inkjet printer 623
Input
 port 30
 unit 10
Instruction
 cycle 22, 35, 67, 108, 244
 fetch 244
 format 22, 114
 hazard 676
 level parallelism 649, 650, 746
 pipeline 658
 pipelining 47, 657
 pre-fetch 45, 336, 337
 queue 336
 sequencing 99
 set 6, 118

- set design 108
 - types 100, 118
 - level parallelism 745
- Instructions 6
- Integer 140
- Integer arithmetic 140
- Integrated drive electronics 428
- INTEL 80486 696
- Intel Pentium-pro 730
- Intelligent controller 528
- Interface adapter 528
- Interleaving 68, 323
- Internal
 - bus 34
 - interrupts 467
- Interrupt 28, 466
 - concept 466
 - handling 466
 - mode 483, 484
 - nesting 474
 - priority 475
 - sensing 469
 - sequence 478
 - types 467
- Invalidation 356
- IP cores 647
- ISA bus 556
- Isochronous data transfer 555
- Isochronous transfer 590
- Itanium 753, 755

J

- Job level parallelism 650
- Joystick 631
- JUMP 166

K

- Key register 379
- Keyboard 603
 - driver 532
 - function 603
 - interface 531, 532, 534
 - organization 606

- Keypad 603

L

- Landing zone 408, 415
- Lands 447
- Laser printer 619, 420
- Latency 300, 323, 414
- Least Frequently Used (LFU) algorithm 354
- Least Recently Used (LRU) 354
- LED printer 622
- Level 1 cache 362
- Level 2 cache 362
- Light Pen 631
- Line printer 616
- Linear
 - address 375, 376
 - address space 375
 - motor 413
 - serpentine recording 438
- Little-endian assignment 117
- LOAD 166
- Local
 - bus 522, 525
 - storage 163
- Logical
 - address space 375
 - shift 96
- Longitudinal recording 388
- Loop Unrolling 705, 706
- Loosely coupled multi-processor 776, 777
- LQP 616

M

- Machine
 - check 792, 244
 - language 6
- Macro 121
- Macro-operation 27
- Magnetic
 - core memory 304
 - drum 387

- optic drives 626
 - tape 437
- Main memory 13, 302
- Mainframe 37, 161, 250
- Mantissa 143
- Mark 73
- Maskable interrupts 479
- Master 508
- Mechanical keyswitch 605
- Membrane keyswitch 606
- Memory 10
 - addressability 21
 - allocation 327
 - bandwidth 338
 - block 343
 - bus 500, 519
 - hierarchy 329
 - Interface 162
 - interleaving 46, 338, 341
 - mapped I/O 327, 462
 - order Buffer 733
 - read 162, 504
 - space 327
 - write 163, 505
- MESI Protocol 786
- Message
 - phase 582
 - pipe 590
- Mezzanine bus 524, 525
- MFM 389
- Micro-operations 27
- Micro-ops 730
- Microcomputer 37
- Microprogrammed control unit 67
- MIMD 653
- Minicomputer 37, 250
- MIPS 40
- Mirroring 432
- MISD 653
- MODEM 546, 628
- Motherboard 51, 52, 524
- Motorola 68010 515
- Mouse 627

- Multi-cycle Design 251
- Multi-function devices 563
- Multi-level Cache 362
- Multi-processor system 775
- Multicycle design 252
- Multiphase clocking 252
- Multiple Bus 517, 572
- Multiplex mode 495
- Multiplication 207
- Multiport memory structure 779

N

- Name dependency 709
- Nested interrupts 29
- Network-attached storage 436
- NLQ printer 616
- NMI 476
- Non-impact printer 616
- Non-maskable interrupt 476
- Non-restoring division 220
- NOOP 166
- Normalization 144
- North bridge 524, 572
- Null modem 551
- NUMA multi-processor 777
- Number systems 5

O

- On-chip cache memory 330
- Open-loop positioning system 409
- Open-loop system 409
- Operand fetch 246
- Operand forwarding 671
- Operating systems 8, 36
- Operator interrupt 468
- Optical disk storage 446
- Optimizing compiler 48
- OUT instruction 30, 461
- Out-of-order execution 700, 701, 708, 730, 734, 752
- Output
 - dependency 702
 - unit 10

Overflow 190
Overlap 648, 649

P

Page
 description language 619
 directory 376
 fault 368
 table 376
Paging 370, 376
Parallel
 adder 194
 interface 458
Parallelism 649, 775
Parity bit 548
Parking zone 408
PCI
 bridge 557
 bus 524, 525, 556
 bus cycle 564
 configuration mechanism 569
 express 557, 570, 571
PCI-X 557, 570
Pentium 696
Pentium 4 736
Pentium processor 697
Pentium-pro 731
performance 648
Peripheral devices 12, 603
Perpendicular magnetic recording 388
Physical address space 375
PIC 476
PING 592
Pipe stage 658
Pipeline 658, 669, 670
 bubbles 670
 design 664
 efficiency 662
 hazards 668
 latency 661, 675
 overhead 664
 performance 663
 register 668
 repeat rate 662

 processor 653, 657
Pipelining 657
Pits 447
Plan of Execution 746
Platter 404
Plotter 630
Plug and play 559, 585
Polling 482, 498
Portable computers 37
Power fail interrupt 468
PowerPC 647, 739
PPD 325
Pre-decode 46
Pre-shipment burn-in 787
Predication 754
Prefetch buffers 336
Primary
 cache 362
 memory 13
Printer 613
 characteristics 614
 controller 537, 539, 540
 functions 613
 types 614
Processor
 design 93
 level parallelism 649, 650, 651
Product ID 559
Program 5, 6
 counter 25
 exception 190
 level parallelism 650
 memory 16
Programmed mode 481
PROM 306
Protected
 mode 372, 375
 virtual address mode 371
Pseudo Static RAM 321

R

RAID 431, 432
Rambus RDRAM 325
RAMDAC 635

- Random
 - access memory 18, 301
 - choice algorithm 354
- Raster scan 608
- RAW Hazard 672
- Read Only Memory 18
- Read/write head 390, 394
- Ready 505
- Real address mode 371, 375
- Real number 140, 143
- Recoding 211
- Recording methods 438
- Redundant Array of Inexpensive Disks 431
- Refresh 314, 319
- Register 12, 309, 560
 - addressing 127
 - architecture 64
 - file 163
 - indirect addressing 127
 - renaming 701, 702, 709, 717, 734
 - transfer 91
 - windows 646
- Alias table 730, 734
- Registers based CPU 110
- Relative addressing 128
- Reorder buffer 709, 710, 734
- Replacement algorithm 345, 353
- Reselection phase 580
- Reservation station 718, 720
- Reset
 - sequence 242
 - vector 242
- Restoring division 217
- Retirement register 736
- Retirement unit 700
- RIMM 327
- Ripple Carry Adder 194
- RISC 47, 105, 107, 108, 641, 648
- RISC architecture 641
- RISC processors 643, 644, 646, 648
- RLL 391
- Root hub 590
- Rotational delay 414
- Rounding 229

- RS 232C Interface 545, 549
- Run Length-Limited 391

S

- Scalar
 - addition 761
 - processor 653, 760
 - registers 763
- Scancode 532
- Scanner 629
- Scanning method 439
- Scatter operation 764
- Scoreboard 710, 657, 712
- SCSI 429, 573
 - bus phases 577
 - bus signals 576
 - controller 427
 - ID 575
 - interface 427
- SCSI-2 574
- SCSI-3 574
- SDRAM 322, 323
- Second Generation Computers 80
- Secondary
 - cache 362
 - memory 302, 330
 - storage 13
- Sector
 - buffer 426
 - interleaving 399
- Sectors 394
- Seek time 414
- Segment
 - descriptor 374
 - descriptor cache 374
 - selector 371
- Segmentation 370, 371, 373
- Selection Phase 580
- Selective Masking 475
- Selector 373
- Self-test 532
- Semiconductor memory 304
- Semirandom access memory 301

- Sequential
 - access memory 301
 - ALU 157
 - circuits 250
- Serial
 - communication 546
 - interface 545, 549 458
- Serpentine pattern 442
- Server systems 775, 792
- Service processor 795
- Set Associative Mapping 351, 352
- Shared memory multiprocessor 776
- Shift register 156
- Shoe-shining effect 439
- Sign Extension 188
- Signals 502
- Signed magnitude form 141
- SIMD 651
- Single Bus concept 517
- Single-cycle design 252
- SISD 651
- Slave 496, 508
- Small Computer System Interface 429
- Snooping 356, 357
 - technique 784
- Snoopy cache controller 784
- Soft Sectoring 396
- Software 7
 - fault 788
 - interrupt 469
 - line feed 539
 - pipelining 707
 - RAID 436
- South bridge 524, 572
- SPARC 646, 647, 740
- Spatial locality 343
- SPD 325
- SPEC 41, 42
 - groups 42
 - rating 42
 - suite 43
- Speculative execution 704, 705, 707, 733, 735
- Spindle motor 404, 413
- Split Cache 361, 370
- Splitter 636
- Stack
 - addressing 130
 - architecture 64
 - based CPU 111, 113
 - concept 111
- Static
 - memory 305
 - scheduling 675, 707, 708, 745, 746
- Status
 - change interrupt 468
 - flags 103
 - signals 529, 536
- Stepper motor 393, 404, 409
- Stop bit 548
- STORE 168
- Stored
 - program computer 73
 - program concept 12
- Straddle erase 404
- Stream pipe 590
- Stride 764
- Striping 793
- STROBE 512
- Structural hazard 668, 669
- Subroutine 122
- Subtraction 205
- Sun server families 796
- Super pipeline 664
- Superscalar
 - architecture 47, 691, 697
 - pipeline 692
 - processor 653, 700, 749
- Swapping 370
- Symmetric multi-processor 777
- Synchronization 250
- Synchronous
 - communication 547
 - data transfer 582
 - DRAM 322
 - interface 20
 - transfer 508, 510, 511
- System
 - bus 455, 518, 525
 - interface 14

- program 9
- software 69

T

- TAG 344, 346
- Target 429, 560, 564, 566, 573
- Temporal locality 343
- The QIC standard 442
- Thermal printers 619
- Thin film
 - heads 406
 - inductive 406
- Thornton technique 710
- Throughput 40
- Tightly coupled multi-processor 776
- Token 591
- Packet 592
- Tomasulo's algorithm 717, 718
- Toner 622
- Touch pad 632
- Trace
 - cache 736
 - scheduling 707, 748
- Trackball 628
- Transactional buffer 733
- Translation lookaside buffer 377
- Transverse scan 439
- Tree instruction 751
- Truncation 229
- TTL video interface 609
- Tunnel erase 404
- Two's complement 141
- Typematic action 532, 607

U

- UART 551
- UMA multi-processor 777
- Unconditional branch instruction 676, 677
- Unified cache 361, 364
- UNIVAC 80
- Universal bus concept 501
- Universal Serial Bus 554
- USB 554, 585, 586, 587, 589

- USB 2.0 587
- USB 3.0 587
- USB Cable 588
- USB On-The-Go 586
- USB Packets 590, 591
- USB1.0 586
- USB1.1 586
- USB2.0 586
- USB3.0 586

V

- Vector 759
 - code 472, 473, 478
 - computing 759, 760, 766
 - instruction 759
 - processing 760
 - processor 653, 759, 760, 764, 769
 - registers 763
 - table 473
- Vendor ID 560
- VESA bus 523
- Video
 - bandwidth 612
 - buffer 327
- Virtual
 - 8086 mode 375
 - memory 303, 368, 369
 - memory interrupt 368
 - memory mechanism 373
 - pipes 589
- VLIW 746
 - architecture 745, 752
 - compiler 748
 - processor 745, 749
- Volume-drive spanning 793
- Von Neumann concept 12
- Von Neumann Machine 77
- VRAM 634
- VSYNC 608

W

- Wait state 506, 507
- WAR Hazard 674
- WAW Hazard 674

Winchester

hard disk 408, 413

technology 408

Wireless

keyboard 603

USB 586

Word length 17

Write

buffer 341, 342

invalidate protocol 785

protect 396

update protocol 785

Write-back 356

Write-through policy 355

Z

Zero wait state 507

Zip drive 626